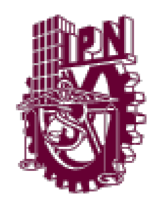


Instituto Politécnico Nacional

Unidad Interdisciplinaria de Ingeniería
y Ciencias Sociales y Administrativas



SISTEMA AUTOMATIZADO PARA EL CONTROL DE PROYECTOS EN LA PEQUEÑA Y MEDIANA EMPRESAS MEXICANAS

(primera etapa)

Registro CGPI 20060083
ANEXO AL INFORME FINAL

Profesores Participantes:
Ing. Carlos Careaga De La Garza
Lic. Humberto Oviedo Galdeano
Ing. Mario Oviedo Galdeano

México D.F., enero de 2006

INTRODUCCION

El paquete se diseñó para obtener tres productos de software que pudieran operar independientemente o bien integrados en un solo paquete y con otros módulos que se podrán añadir en la segunda etapa del proyecto prevista para un siguiente período. Los tres productos se han denominado ***SysRed, SysCpm y SysPert.*** Nuestro paquete de software pretende automatizar la construcción de una red de actividades y utilizarla para la programación de tiempos en cualquier proyecto. Su diseño se basa en las técnicas de planeación y programación.

Por limitaciones de espacio para este anexo de la versión electrónica del informe final, se describe únicamente la base metodológica del programa SysPert que es el más completo porque utiliza prácticamente todos los módulos, pero en el manual técnico de cada programa se describe ampliamente la tácnica de planeación y programación empleada para su desarrollo.

La metodología PERT cuyo modelo considera en forma más realista, el concepto de probabilidad para los tiempos de las actividades que intervienen en un proyecto, las duraciones de los trabajos son tiempos esperados a cumplir. El método PERT probabilístico se basa en tiempos estimados de gente que conoce el tipo de actividad por realizar. En este anexo se hace una explicación general de la técnica que se aplica para resolver problemas de redes de actividades en proyectos, con una descripción del algoritmo original.

En la segunda parte se muestra el diseño del paquete y se hace una descripción de todos los módulos que lo constituyen, con una explicación detallada de sus partes relevantes.

Se incluyen algunas pantallas representativas de la interfaz gráfica del usuario y una breve descripción de su operación. Se detallan las estructuras de datos empleadas.

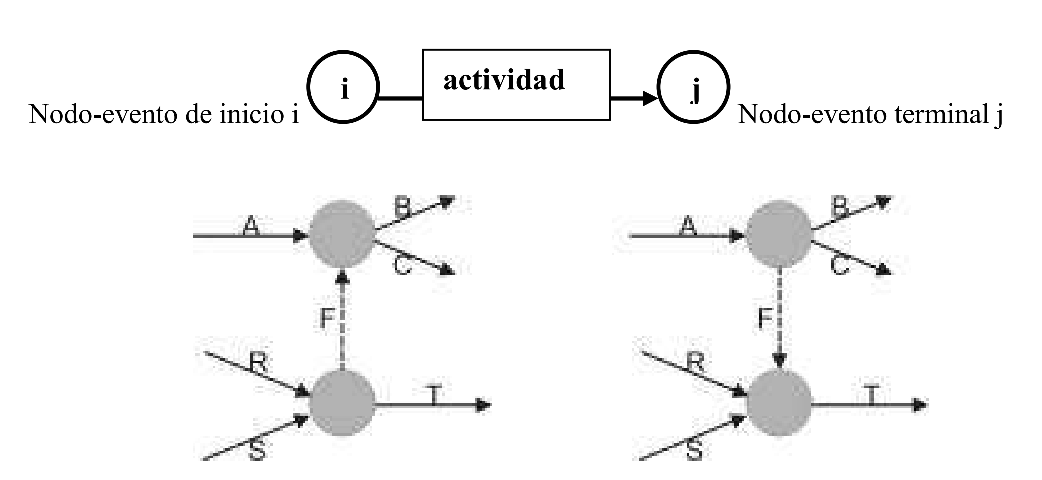
En el apéndice se presentan los listados completos y se agrega un diccionario de datos con referencias cruzadas.

El paquete fue desarrollado con la herramienta Delphi versión 5 cuya licencia académica pertenece a la UPIICSA.

TECNICAS DE PLANEACIÓN Y PROGRAMACIÓN

El método PERT, utiliza la red que facilita seguir el trabajo, ilustrando su secuencia; también se usa para programar tiempos esperados de actividades y de eventos-nodos con la probabilidad de una distribución normal para cumplirlos. Un proyecto combina actividades interrelacionadas conforme al tiempo originando una secuencia lógica tal, que empezar un trabajo (excepto los iniciales) depende de que otros se terminen.
En general, **diseñar la red-proyecto puede ser una labor difícil**, el alumno o el usuario requiere de imaginación e inventiva; primero se hace una lista de los distintos trabajos por hacer, decidiendo la secuencia (actividades precedentes) entre ellos y luego se intenta el modelo de red. **La unión de diversos nodos y flechas forma una red con distintas rutas** que conectan los trabajos de inicio con los intermedios y termina con las actividades finales del proyecto.

Una actividad en flecha (i, j), debe tener en ambos extremos, un nodo i (i =1, 2,...) de inicio y un nodo j (j =1, 2,...) terminal; **el par (i, j)** en donde **i < j, identifica cada trabajo**. Es importante el diseño de una **red-proyecto** pues puede ser difícil de representar si se busca **evitar cruces de líneas**, ubicar los nodos de inicio y terminación en la orilla de la red, y **usar sólo las necesarias actividades ficticias;** estas sirven para completar las precedencias entre trabajos reales. Los siguientes esquemas son ejemplo para representar e interpretar la posible relación de precedencia entre actividades de cualquier red proyecto:



Ejemplo de un trabajo **ficticio F**, su efecto en una secuencia de trabajos reales en un proyecto.

A la izquierda de la figura se muestra que los trabajos B y C se inician hasta que su precedente A termina; además, según la **flecha ficticia F**, también deben esperar el fin de otras precedentes R y S; el inicio de T sólo depende de R y S. En la derecha de la misma figura, se invierte la ficticia, continúa la secuencia en B y C después de A, pero T depende de las precedentes R, S y A.

La regla para utilizar un trabajo ficticio es evitar que dos o más actividades se identifiquen con un mismo par (i, j), esto es, pueden tener el mismo nodo (i), pero con diferente (j), o bien, el mismo (j) pero con diferente (i). Para corregirlo se agregan, una flecha F y un nodo (k) ficticios. Un trabajo-flecha **ficticia, no consume tiempo**, pero este tipo de red la usa al representar precedencias.

MÉTODO PERT PROBABILISTICO.

Mediante el concepto de probabilidad, la metodología PERT tiene un modelo que maneja en forma más realista los tiempos de las actividades que intervienen en un proyecto. En tal caso las duraciones de los trabajos son tiempos esperados a cumplir. El método PERT probabilístico se basa en tiempos estimados de gente que conoce el tipo de actividad por realizar; así toma en cuenta la opinión de expertos que pueden proporcionar valores de duración de la siguiente manera:

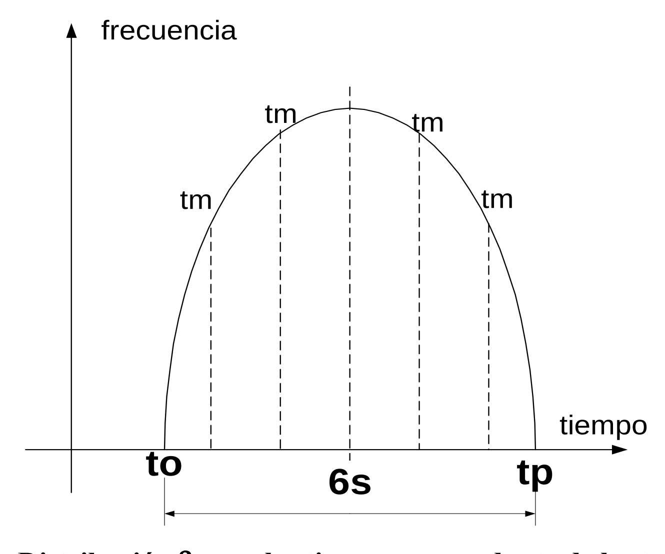
Tres estimaciones para la duración de cualquier trabajo del proyecto:
Tiempos esperados, variancias y programación.

t o = tiempo **optimista**, si todo transcurre bien.
t m = tiempo **más probable**, si todo transcurre ordinariamente
t p = tiempo **pesimista**, si todo transcurre mal.

Luego se aplican en la fórmula del PERT para calcular el tiempo esperados **te** como promedio ponderado:

$$t_e = \frac{t_o + 4t_m + t_p}{6}$$

Suposiciones del modelo PERT probabilístico.- Los tiempos esperados **te** de los diversos trabajos se consideran como **variables aleatorias** de una distribución de probabilidad **β de** tipo gaussiana**;** los valores de variable aleatoria se dispersan dentro del espacio de tres desviaciones, a uno y otro lado, de la **media**; los valores **te** son estadísticamente **independientes.**



Distribución β para los tiempos esperados te de los trabajos (i , j)

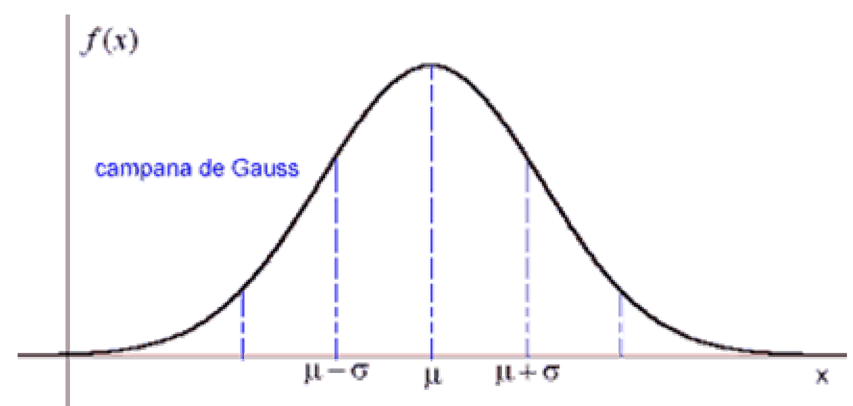
La medida de dispersión para la variable aleatoria **te** se calcula con la fórmula para la variancia siguiente:

$$S_{te}^2 = \left[\frac{t_p - t_o}{6}\right]^2$$

La distribución β es gaussiana pero se distingue de otras distribuciones de este tipo porque las colas de la campana cruzan el eje horizontal, teniendo de esta manera la ubicación para las duraciones pesimista y optimista, las cuales no pueden rebasar dichos valores ya sea para más o para menos. La ubicación para el tiempo más probable puede ser en cualquier lugar del intervalo entre t o y t p que se consideran valores extremos para la

variable aleatoria **t e**. Para el intervalo entre t o y t p se consideran 6 desviaciones. La fórmula para el tiempo esperado proporciona la ponderación de las estimaciones de los expertos para calcular el mismo; así se puede observar la ponderación de 2/3 partes de área bajo la campana para el tiempo más probable y la tercera parte complementaria del área bajo la campana, se reparte igualmente con un sexto del área para cada uno de los tiempos optimista y pesimista.

El método PERT probabilístico también considera **tiempos esperados Ti** para cada uno de los **eventos-nodos (i)** del proyecto, pero distribuidos conforme a la **distribución normal**; en tal caso, el tiempo esperado Ti se calcula con la sumatoria de los tiempos esperados correspondientes a los trabajos que forman la **ruta mayor** para cumplir dicho evento-nodo (i) : $\sum te = Ti$



Distribución Normal para tiempos esperados Ti de los eventos-nodos
La suma de variancias en tal ruta mayor resulta en variancia de Ti: $(Si)^2 = \sum (Ste)^2$

Entre todas las rutas de la red, la que resulta **estadísticamente más larga, es crítica**; los diferentes nodos eventos del proyecto se cumplen en un tiempo $Ti = \sum te$, (suma de valores aleatorios te). Esta variable Ti también es aleatoria pero distribuida conforme a una **normal**.

Los valores obtenidos para cada uno de los eventos del proyecto son particulares, lo cual es prácticamente imposible manejar. Para superar este inconveniente, el modelo PERT probabilístico propone una conversión de los valores particulares, hacia los valores de una distribución **normal estándar** con parámetros:
$N(\mu, \sigma^2) = N(0, 1)$

Y con la fórmula :

$$Zi = \frac{(TP)i - Ti}{Si}$$

En donde:
Z_i es la desviación normal estándar que se puede encontrar en las tablas de distribución normal.
T_i es el tiempo esperado para el evento i de interés en el proyecto.

S_i es la raíz cuadrada de la variancia (S_i)^2 que se puede calcular por la suma de las variancias de los trabajos sobre la ruta mayor para cumplir el evento **i**
(T P) i es **tiempo programado por la administración** del proyecto para un **evento i** del proyecto.

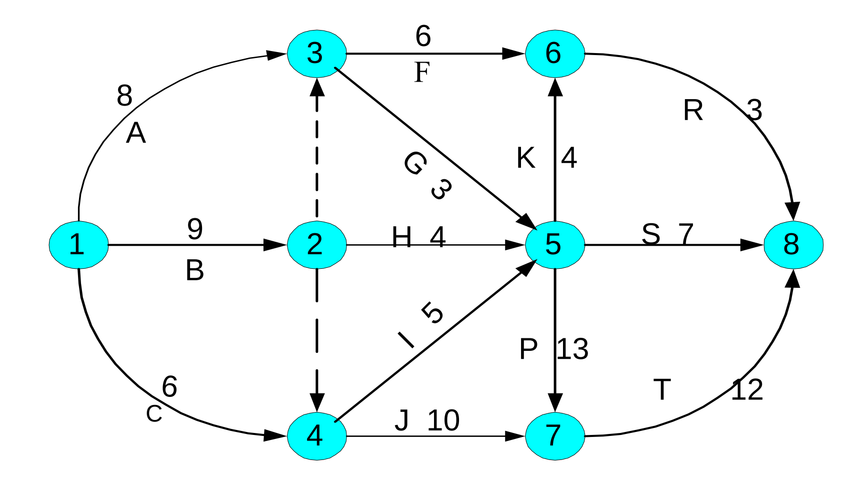
De esta manera, el gerente del proyecto puede programar con la probabilidad de que un cierto evento nodo i se cumpla cuando mucho, en un tiempo (TP)i que se cuantifica en forma conveniente de acuerdo al modelo PERT probabilístico. Estos valores de probabilidad se pueden tener consultando las tablas de distribución normal estándar extensamente publicadas.

Ejemplo.- Con la red proyecto, cuyos tiempos estimados se anotan en tabla como (t o, t m, t p) para cada trabajo, determine un **programa de tiempos esperados T i** para cada evento-nodo **i**.

Trabajo (i, j)	Duración estimada			Tiempo esperado Cálculo con la fórmula del **te**	Variancia Cálculo con la fórmula de S^2 **i**
	to	tm	tp		
A	7	8	9	8	1/9
B	5	7	21	9	64/9
C	4	6	8	6	4/9
F	3	5	13	6	25/9
G	1	2	9	3	16/9
H	2	2	14	4	4
I	3	5	7	5	4/9
J	9	10	11	10	1/9
K	2	4	6	4	4/9
P	10	13	16	13	1
R	2	3	4	3	1/9
S	5	7	9	7	4/9
T	9	12	15	12	1

Tiempo esperado y variancia de trabajos en red proyecto del ejemplo

El software produce la solución de nodos en pares (i, j), de este ejemplo con 13 trabajos.



Red-proyecto del ejemplo, con trabajos y tiempos te.

La aplicación del software al ejemplo planteado, entrega los siguientes resultados como programa de tiempos a cumplir en el proyecto con la información de las holguras en trabajos no críticos. Así la administración, puede preparar el movimiento de recursos a la actividad que le convenga. La tabla que sigue contiene la estructura que se preparó para la interfase de entrega del programa método PERT para proyectos.

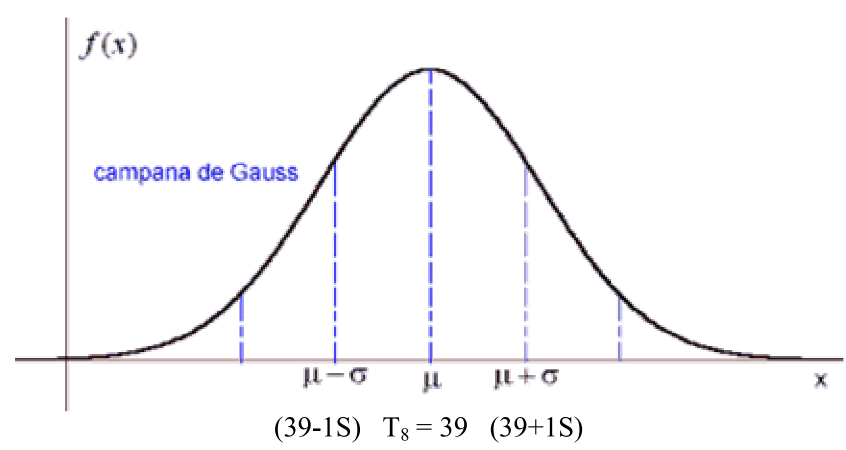
NODO-EVENTO i	Ruta mayor	Tiempo esperadoTi	Varianza S^2i	**tiempo programado (TP)i**	Zi={(TP)i-Ti}/Si	**Probabilidad**
1	-	0	0	-	-	-
2	1,2	9	64/9	9	0	0.5
3	1,2,3	9	64/9	8	-0.375	0.3538
4	1,2,4	9	64/9	10	0.375	0.6461
5	1,2,4,5	14	68/9	13 15	-0.363 0.363	0.358 0.642
6	1,2,4,5,6	18	72/9	17 18	-0.353 0	0.362 0.5
7	1,2,4,5,7	27	77/9	25 26 28	-0.683 -0.341 0.341	0.2473 0.3665 0.6334
8	1,2,4,5,7,8	39	86/9	37 38 40	-0.646 -0.323 0.323	0.2591 0.3733 0.6266

Programa del ejemplo, con probabilidades para los tiempos (TP)i en los nodos-evento i.

Los tiempos de la gerencia **(TP)i son supuestos,** que se pueden programar para cada **evento-nodo i**.

Ejemplo (a).- Con la misma red proyecto ejemplo y tiempo esperado $\mathbf{T_8 = 39}$, determine lo siguiente: Calcule la probabilidad de que el proyecto **se complete en el tiempo esperado** $T_8 = 39$, pero dentro de **1 desviación estándar**.

$\mathbf{Z_8 = \{(TP)_8 - T_8\} / S_8}$; **pero:** $\mathbf{(TP)_8 = \{T_8 \pm 1S_8\} \Rightarrow Z_8 = \{T_8 \pm 1S_8 - T_8\} / S_8 = \pm S_8 / S_8 = \pm 1}$

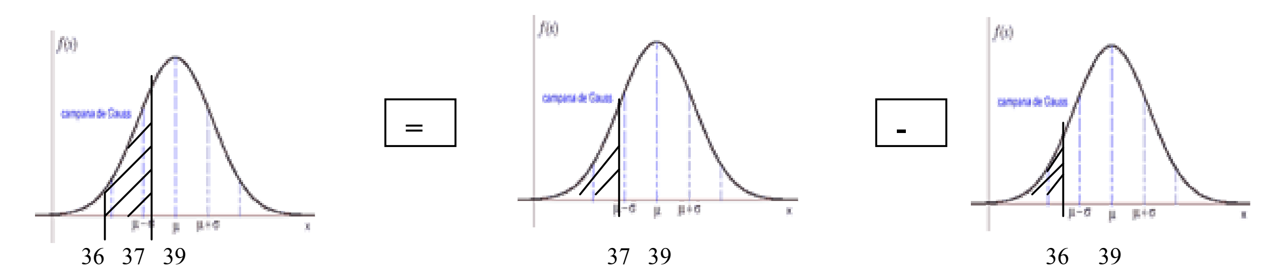


(39-1S) $T_8 = 39$ (39+1S)

$\mathbf{P\{T_8 \pm 1S_8\} = 0.84134 - 0.15866 = 0.68268}$ (tablas de distribución Normal para Z = 1 y Z = -1)

Campana de Gauss de distribución normal, ejemplo (a)

Ejemplo (b).- Con el mismo ejemplo calcule la probabilidad de que el proyecto se cumpla exactamente dentro de la unidad de tiempo 37; es decir, que $P\{(TP)_8 = 37\}$.



$$\mathbf{P\{(TP)_8 = 37\} = P\{(TP)_8 \le 37\} - P\{(TP)_8 \le 36\}}$$

$$P\{(TP)_8 = 37\} = P\{Z_8 = (37 - 39) / \sqrt{86/9}\} - P\{Z_8 = (36 - 39) / \sqrt{86/9}\}$$

$$\mathbf{P\{(TP)_8 = 37\}} = P\{Z_8 = -0.7276\} - P\{Z_8 = -0.9704\} = 0.23343 - 0.16592 = \mathbf{0.06751}$$

Figura 9(b). Campanas de Gauss de distribución normal para el ejemplo (b)

Ejemplo (c).- Calcule un intervalo de confianza de 95% para la terminación de proyecto

$\alpha = 1 - 0.95 = 0.05$, $\alpha / 2 = 0.05/2 = 0.025$;

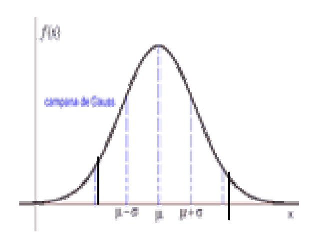
en tablas de distribución normal $\mathbf{Z = \pm 1.96}$

$\pm Z_i = \{(TP)_i - T_i\} / S_i \Rightarrow (TP)_i = T_i \pm Z_i S_i = 39 \pm (1.96)(\sqrt{86/9})$

$\Rightarrow T_i - Z_i S_i \le \{IC\}_{0.95} \le T_i + Z_i S_i$

$39 - (1.96)(\sqrt{86/9}) \le \{IC\}_{0.95} \le 39 + (1.96)(\sqrt{86/9})$

$\mathbf{33 \approx 32.94 \le \{IC\}_{0.95} \le 45.05 \approx 45}$

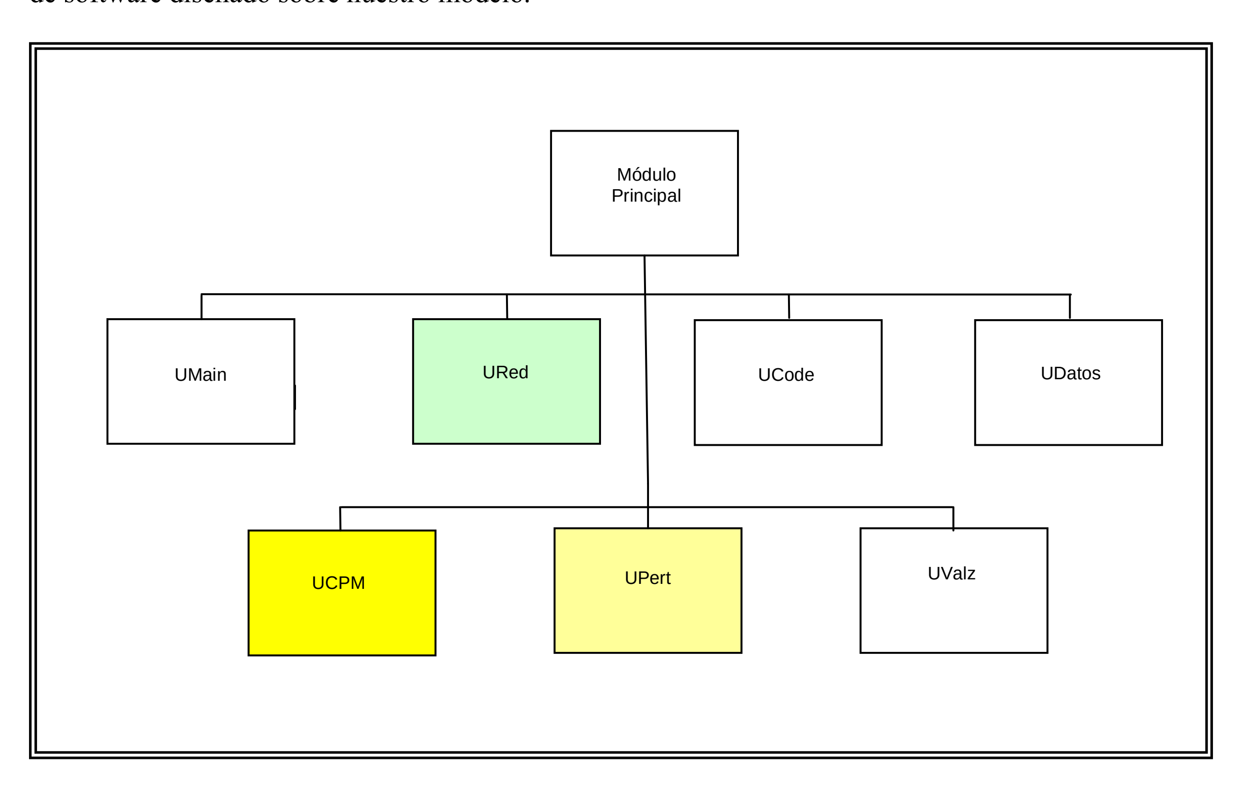


-1. 96 +1. 96

Figura 10(c).- Campana de Gauss de distribución normal del ejemplo (c)

ARQUITECTURA DEL SISTEMA AUTOMATIZADO PARA EL CONTROL DE PROYECTOS EN LA PEQUEÑA Y MEDIANA EMPRESAS MEXICANAS (primera etapa)

El siguiente diagrama de bloques muestra los ocho módulos principales que componen el paquete de software diseñado sobre nuestro modelo.



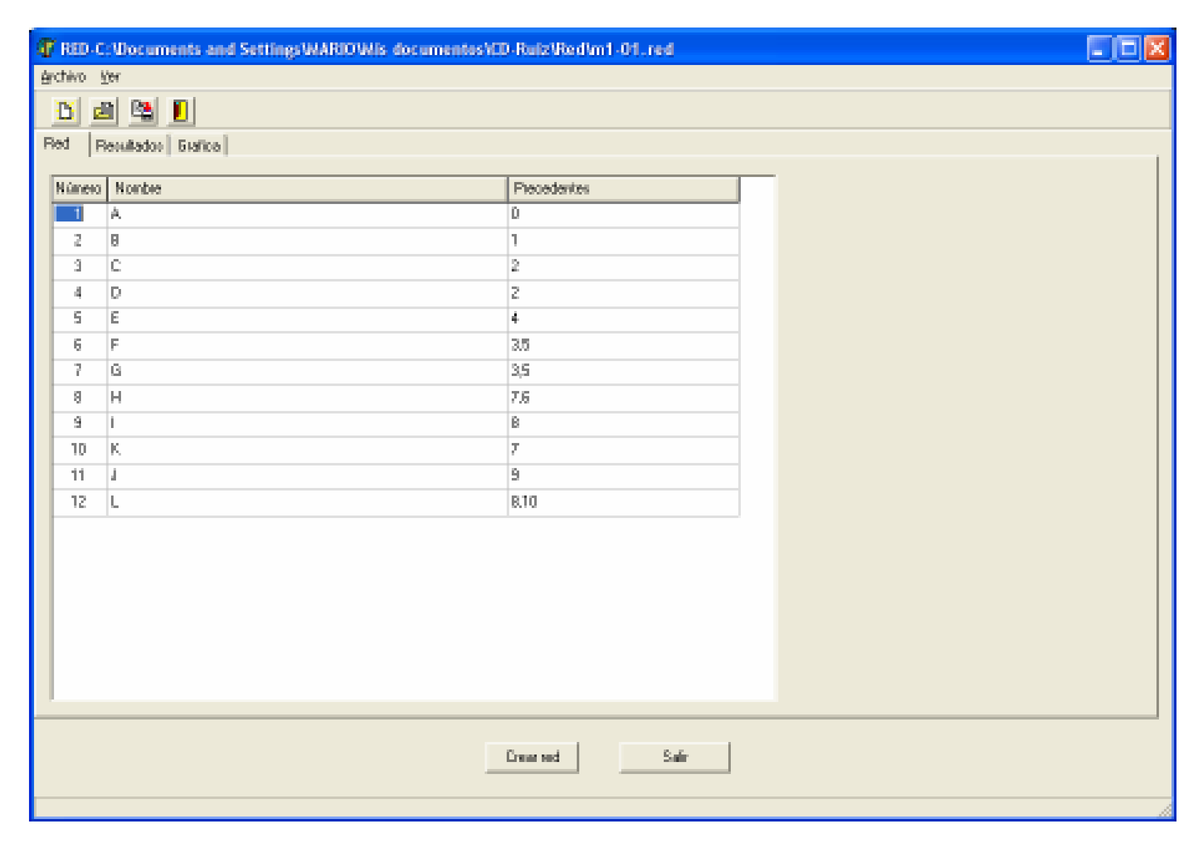
El paquete se diseñó para obtener tres productos de software que pudieran operar independientemente o bien integrando un solo paquete con otros módulos que se podrán añadir en la segunda etapa del proyecto prevista para un siguiente ciclo. Los módulos UCode y UDatos son comunes a todas las versiones. Los módulos Principal y UMain se tienen que configurar de acuerdo a cada versión ya que UMain contiene la interfaz del usuario para cada caso. A continuación se describe el conjunto de módulos que conforman los productos que hemos denominado SysRed, SysCpm y SysPert.

SysRed

Este programa crea una red a partir de la lista de actividades y sus precedentes. El modelo utilizado se denomina AOA (actividades en los arcos). Los módulos que constituyen este programa son Principal, UMain, UDatos, UCode y URed. El módulo Principal inicializa el proceso, crea el formulario y corre la aplicación.

UMain es el módulo básico y opera prácticamente toda la aplicación. Contiene al formulario principal y es la interfaz gráfica del usuario; se utiliza tanto para la captura de datos como para el despliegue de resultados.

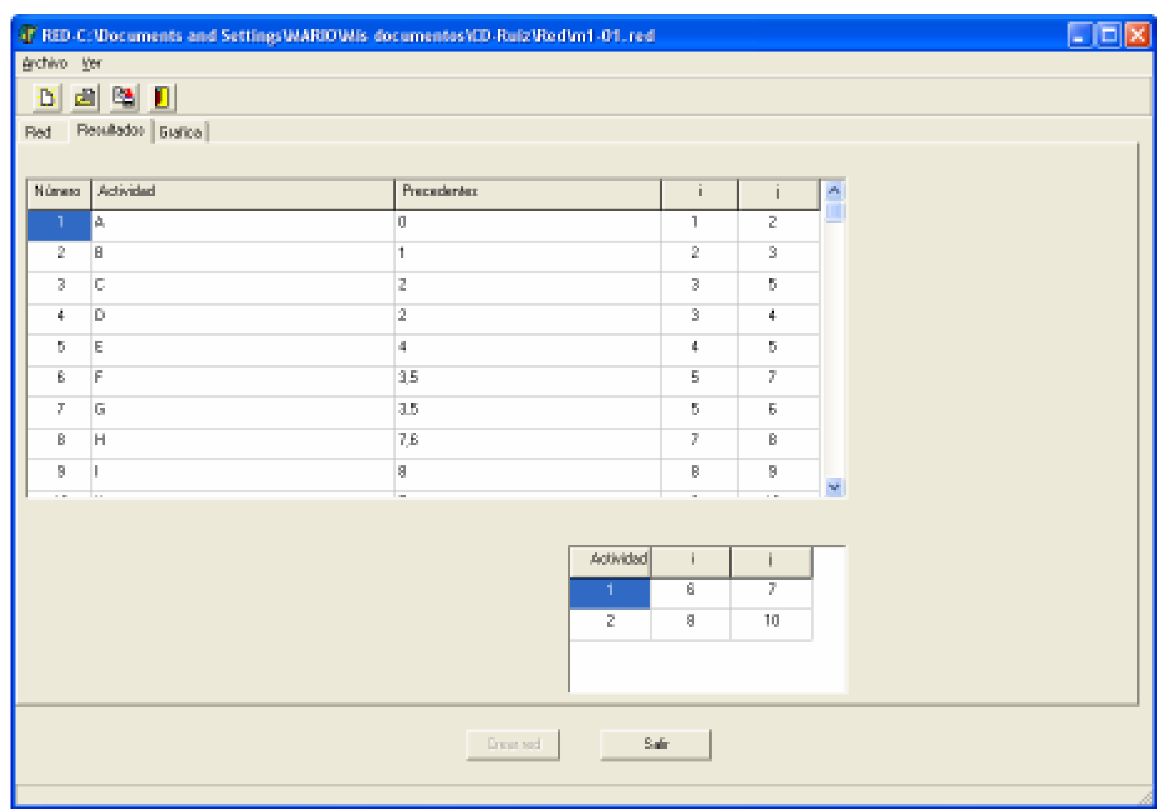
La siguiente figura muestra el formulario con el menú principal, barra de herramientas, la página de entrada de datos (visible), las páginas (no visibles) de resultados en forma tabular y la de la gráfica de la red. La página de entrada muestra la tabla de actividades



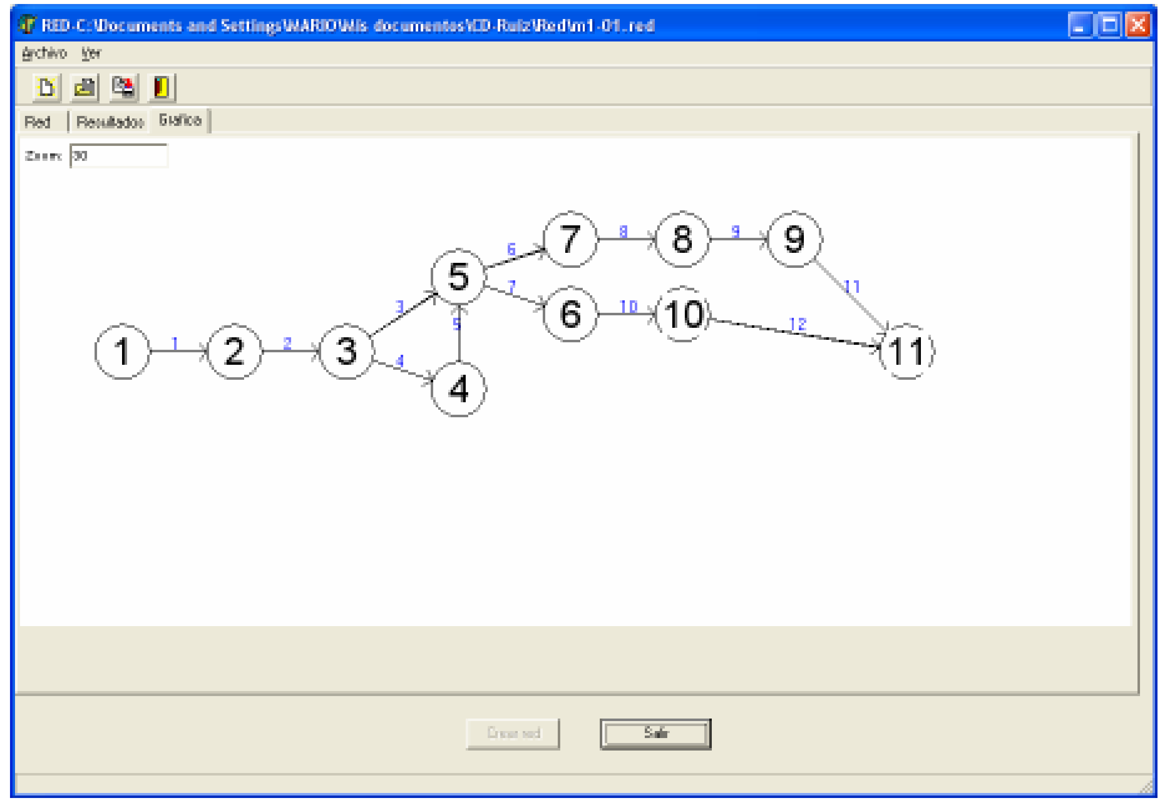
Número	Nombre	Precedentes
1	A	0
2	B	1
3	C	2
4	D	2
5	E	4
6	F	3,5
7	G	3,5
8	H	7,6
9	I	8
10	K	7
11	J	9
12	L	8,10

En la figura aparece la lista de 12 actividades con se vestor de precedentes. La red se obtiene al oprimir el botón rotulado *Crear red* y los resultados se presentan en forma tabular y de manera gráfica.

Las siguientes figuras muestran las dos panatallas de salida: tabular y gráfica.



Salida tabular



Salida gráfica

En la salida tabular se muestran además de los datos, los pares *(i,j)* cuyos valores representan el nodo inicial y el nodo terminal de cada actividad, también aparecen en una tabla aparte las actividades ficticias en caso de haberlas. Por su parte la salida gráfica muestra la red desde su nodo inicial hasta el nodo terminal.

Los procedimientos independientes del formulario son:

1. IniciaFormas: Prepara las tablas de datos y resultados con encabezados y ancho de columnas
2. Resetear: Reinicializa diversos objetos utilizados por la aplicación: vectores y colas
3. MiGrafica: Prepara los datos y crea los elementos para la gráfica de la red
4. CreaGrupos: Crea los grupos de actividades precedentes
5. CargaDatos: Carga el vector VActiva1 y la Cola1 con los valores de la tabla de datos
6. ReCargaDatos: Carga la tabla de resultados con los valores calculados
7. PrepararDatos: Prepara los datos para su proceso asignado códigos a las actividades

Además del formulario, en la unidad UMain se definen las clases TCodigo y aCo cuyos diagramas RE (Representación-Especificaciones) se muestran a continuación.

TCodigo	
Especificaciones procedure VerifComas; function GuardarDatos:integer; procedure CodAparicion; procedure Cm32; procedure CrearMiDondeEstan; procedure GuardaCM32; procedure Ordenar procedure MiOrden; procedure OrdenVP; procedure CR; Function a(X:integer):Integer; procedure borrar;	Representación ElemVp : array of array of integer CodApa : array of array of integer; GruposA : array of array of integer; MiDondeGrupos:array of array of integer; CR1 : array of integer;

aCo

Especificaciones	Representación
destruir;	Tamano:Integer;
Graficar;	Max, Min:integer;
DarTamano;	X:array of integer;
GuardarCordenadas;	Y:array of integer;
ordenar;	LoY:array of array of integer;
MaxMin;	LoXZ:array of array of integer
LasX;	Coor:array of array of integer;
LonY;	CoorTipo:Array of Integer;
Lasy;	CoorTipo2: Array of Integer;
Arreglar;	NNodos:array of Tshape;
CreaObjetos;	Ima:Timage;
NEliminar;	Zoom:TEdit;
NDibujar;	Indice:Integer;
MoverNodo;	Etiqueta:TLabel;
MoverNodo2;	Etiqueta2:array of TLabel
Arrastrar;	BarraHorizontal:TScrollBar;
ArrastraSobre;	BarraVertical:TScrollBar
NMover;	Escala:Integer;
Shift;	Posicion:Integer;
NRevizar	AntBarraHor:Integer;
NRevizar2;	AntBarraVer:Integer;
NBarras;	XIma, Yima:integer
NAcomodar;	Hecha:boolean;
CambiarTamano;	
Flechas;	
QuecarBarras;	
MostrarCombo	
ObtenCoordenadas;	

Al oprimir el botón Crear Red se inician todos los procesos para tener la red y el programa de tiempos. A continuación se muestra el manejador de tal evento.

```
procedure TfmMain.btCreaRedClick(Sender: TObject);
//Inicia los procesos de: preparación de datos, creación de la red, cálculo de tiempos y holguras,
//cálculo de probabilidades y exhibición de resultados
begin
 CargaDatos;
 Preparardatos;
 CargarDatos2;
 {CreaGrupos;}
 ArmaRed;
 Cpm;
 Calcular1.Enabled:=True;
 ReCargaDatos;
 TabSheet2.Visible:=True;
 PageControl1.ActivePage:=TabSheet2;
 btCreaRed.Enabled:=False;
 MiGrafica;
 Grafica.Graficar(PnGraficar);
end;
```

El procedimiento CargaDatos (línea 5) realiza el almacenamiento de los datos básico que identifican a cualquier actividad en el vector objeto VActiva1. La estructura de dichos datos es la siguiente:

```
TVActiva=record
    Index:word;……………..……..….Asigna un secuencial a cada actividad (1..n)
    Nombre:ShortString;……...……Guarda el nombre que el usuario asigne
    LVP:word;………………...……Indica la longitud del vector de precedencias
    VP:ShortString; …...……….......Vector de precedencias de cada actividad
    i,j:word;……………...…………Evento inicial y terminal de cada actividad
    cApar:byte;……………...……..Código de aparición
    cM3:byte;……………...……….Código Modelo 3
    cR:word;……...………........…..Código de repetición
    Ok:byte;………………...………Indica si una actividad ha sido procesada
    Index2:word..............................Indice secundario
   end;
```

y el objeto Cola1 que almacena los índices de todas las actividades, para su proceso:

```
TActiva=record
    Index:word;……………………..Indice de cada actividad
   end;
```

Las siguientes figuras muestran el proceso de todas las atividades y el rol de Cola1 y los vectores VActiva1 y VEstima. El procedimiento CargarDatos2 realiza la carga de datos en VEstima, cuya estructura se muestra a continuación:

```
TEstima= record
    Index:word;
    tOp,
    tMe,
    tPe,
    DurMe,
    Varianza:real;
   end;
```

La siguiente figura muestra los campos de la tabla de datos que se guardan en VActiva1, los que se guardan en VEstima y el índice en Cola1. También se muestra el módulo Codigos produciendo los valores de cAp, CM3 y Cr que se guardan en VActiva1.

PREPARACION Y CARGA

Tabla de datos

Index	Nombre	Precedentes	T. Opt	T.Med	T Pes
1					
2					
3					
--	----------	----	--	--	--
n					

Cola1

1	2	3	4	---	n

VEstima

Index	tOp	tMed	tPes	DurMed	Varianza
1					
2					
3					
--	--	--	--	--	--
n					

VActiva

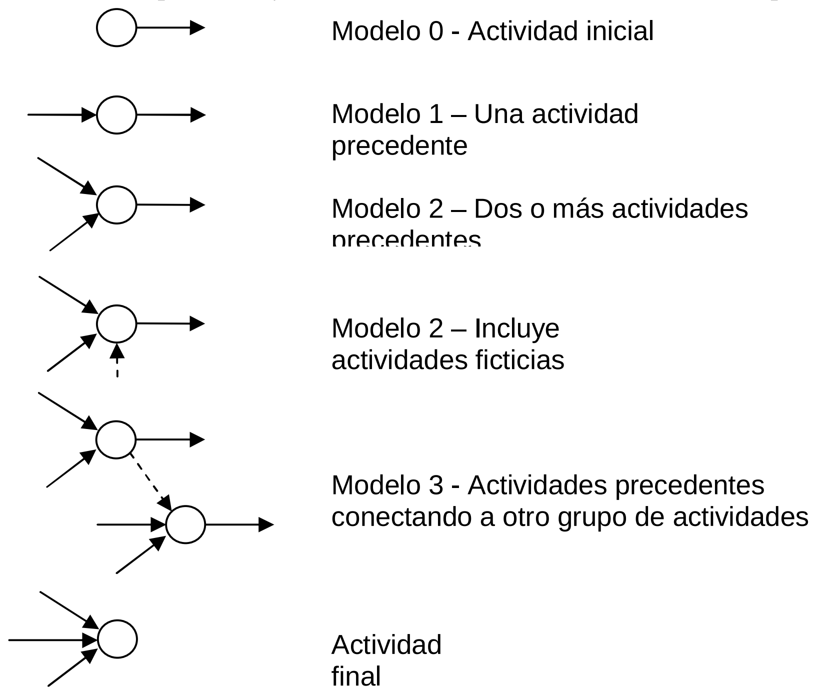
Index	Nombre	LV	VP	i	j	cApar	cM	cR	Ok	Index2
1										
2										
3										
--	-------	--	-	-	-	---	--	-	-	--
n										

Código

El proceso principal es de carrusel, Cola 1 contiene en formación secuencial a las actividades que formarán la red. El índice de cada actividad se saca de Cola1 para su proceso y dependiendo de sus actividades precedentes el proceso pudiera quedar incompleto o concluirse. En el primer caso, proceso inconcluso, la actividad no queda totalmente definida y su índice es regrasado al final de Cola1 hasta que se pueda completar su proceso. Se considera el proceso concluido cuando se han definido los valores de *i* y de *j* que componen el par de eventos inicial y terminal de cada actividad. El proceso encargado de generar los códigos para VActiva1 y que serán la base para crear la red se muestra a continuación

```
procedure preparardatos;
//Prepara los datos para su proceso asignando códigos a las actividades
begin
MiCodigos:=Tcodigos.Create;
with MiCodigos do
begin
 VerifComas;
 GuardarDatos;
 CodAparicion;
 Cm32;
 GuardaCM32;
 MiOrden;
 CrearMiDondeEstan;
 OrdenVP;
 CR;
 Borrar;
end; //with MiCodigos
end; //preparardatos
```

Los algoritmos CodAparaicion y Cm32 se basan en los modelos diseñados para nuestro proyecto.



La red se construye de acuerdo con los modelos de nodos.
Las actividades que llegan tienen códigos asociados al modelo del nodo

Con los códigos asignados a las actividades y definidos los grupos, los algoritmos del módulo URed calculan los pares que hemos denominado conjuntos *(i,j)* correspondientes a los eventos de inicio y terminación de las actividades para la creación de la red.

La implementación en Pascal del proceso en carrusel descrito se muestra a continuación. La estructura principal es un ciclo de repetición hasta que Cola1 esté vacía. Esto es, hasta que todas las actividades tengan definidos los pares *(i,j)*. En VActiva1 se guarda un registro de cada actividad con un campo denominado *Ok* cuyos valores son *cero* para las actividades en proceso y *uno* para aquellas cuyo proceso concluyó. El programa examina *Ok* y si su valor es cero, regresa el índice de la actividad al final de Cola1 en caso contrario elimina la actividad del carrusel. Los valores del par *(i,j)* se almacenan en VActiva1.

En la entrada al ciclo (línea 2) se verifica el código de repetición *cR* de cada actividad para evitar repetir innecesariamente parte del proceso ya que ambas actividades tienen el evento inicial *(i)* común. En la línea 4, se obtiene la actividad con inicio común y en la línea 5 se asigna el mismo evento inicial a la actividad con inicio común. Si el código de repetición *cR* es igual a cero el control se transfiere a la línea 20 para iniciar el proceso de la actividad y obtener el par *(i,j)* invocando al procedimiento *Pares.*

La línea 22 verifica si el *Ok* de la actividad procesada es igual a cero en cuyo caso se regresa la actividad a *Cola1* para su proceso posterior. La línea 24 *decola* la siguiente actividad en *Cola1* y se repite el proceso descrito hasta que *Cola1* esté vacía.

```
1   repeat
2    if (A1.cR>0) {actividad con inicio común con otra}
3     then begin
4      VActiva1.Recall(A1.cR,Ap); {se busca en VActiva1 la actividad con inicio común}
5      A1.i:=Ap.i; {asigna el evento de inicio común, misma i}
6      VActiva1.Store(A1.Index,A1);
7      if (Cola1.Items_In<nT)
8      then begin
9      if not(flJMax)
10      then begin
11       inc(JMax);
12       flJMax:=True;
13      end;
14     VActiva1.Recall(Ax.Index,A1);
15     A1.j:=JMax;
16     A1.Ok:=1;
17     VActiva1.Store(A1.Index,A1);
18    end;
19    end
20    else Pares(Ax);
21   VActiva1.Recall(Ax.Index,A1);
22   if A1.Ok=0
23    then Cola1.EnQueue(Ax,SizeOf(Ax));
24   Cola1.DeQueue(Ax,SizeOf(Ax));
25   VActiva1.Recall(Ax.Index,A1);
26  until Cola1.Is_Empty;
```

El procedimiento *Pares* discrimina entre tres situaciones diferentes por el número de actividades precedentes a la actividad en proceso. La variable *LVP* se relaciona con los modelos descritos anteriormente (ver figura la siguiente figura). En la estructura de opción múltiple case *A1.LVP*

selecciona el llamado al procedimiento que resuelve el caso de las actividades iniciales *ParCero*, o al procedimiento para las actividades que solamente les antece una actividad *ParUno*, o el que resuelve para las actividades con dos o más actividades precedentes *ParMas*.

```
case A1.LVP of
   0:ParCero(Ax);
   1:ParUno(Ax);
   else ParMas(Ax);
  end;{case}
```

- ParCero resuelve el Modelo 0
- ParUno resuelve el Modelo1
- ParMas resuelve los demás casos

URed contiene 18 procedimientos y una función:

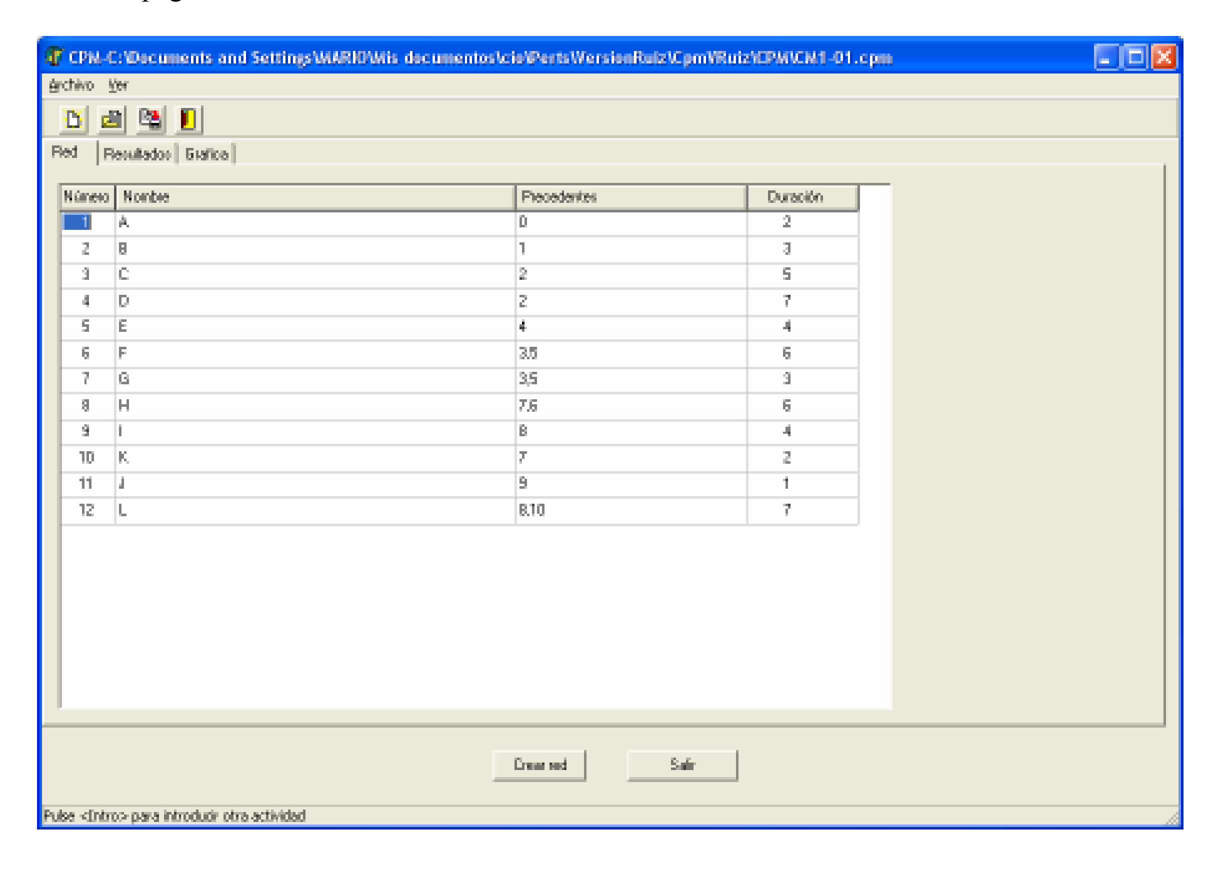
```
procedure ArmaRed;
procedure Caso2;
procedure Caso3;
procedure IDiferente;
procedure IMisma;
procedure jPendiente;
procedure M1;
procedure M1oM2;
procedure M2;
procedure M3;
procedure M3R;
procedure M3S;
procedure ParCero(var Ax:TActiva);
procedure Pares(Ax:TActiva);
procedure ParMas(var Ax:TActiva);
procedure ParUno(var Ax:TActiva);
procedure SolMi(cMi:Queue;Ax:TActiva);
procedure SolPivote(Ap:TActiva);
function gpoVP(gS,VP:ShortString):boolean;
```

SysCpm

Este programa crea una red a partir de la lista de actividades y sus precedentes utilizando el programa *SysRed*. El modelo utilizado se denomina AOA (actividades en los arcos). Los módulos que constituyen este programa son Principal, UMain, UDatos, Ucode, Ured y UCpm. El módulo Principal inicializa el proceso, crea el formulario y corre la aplicación.

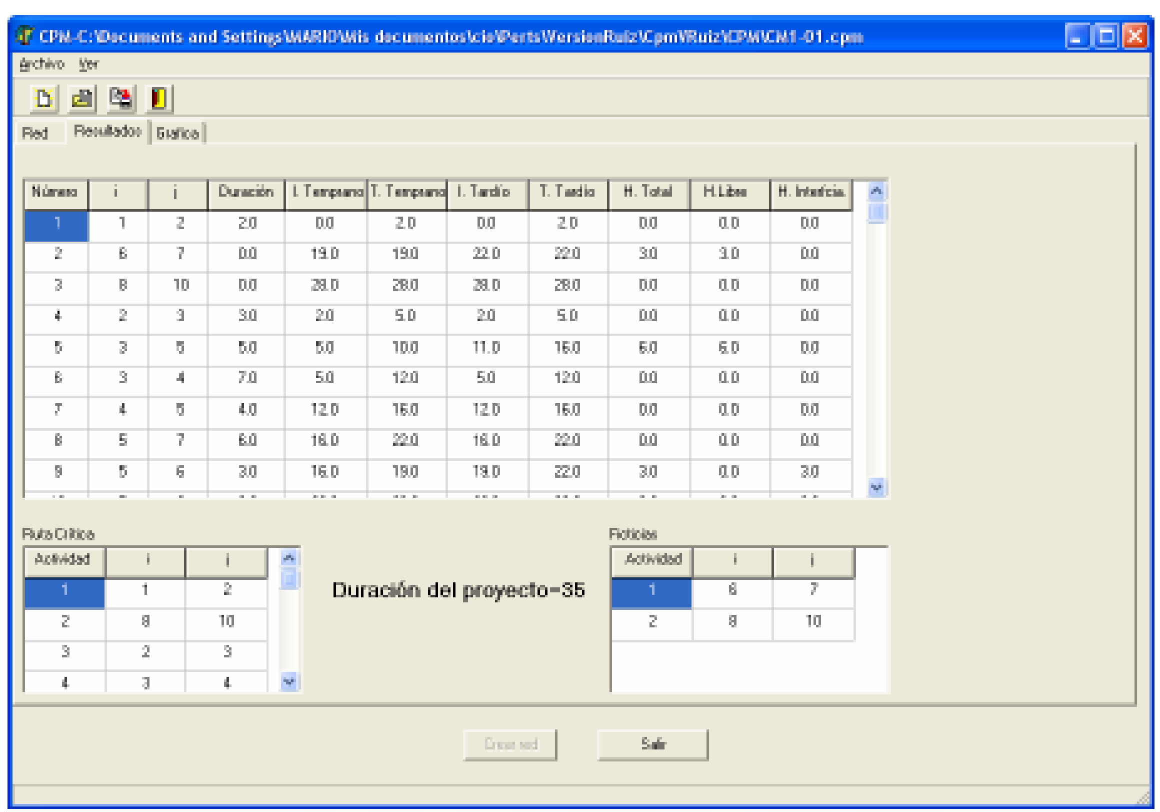
UMain es el módulo básico y opera prácticamente toda la aplicación. Contiene al formulario principal y es la interfaz gráfica del usuario; se utiliza tanto para la captura de datos como para el despliegue de resultados.

La siguiente figura muestra el formulario con el menú principal, barra de herramientas, la página de entrada de datos (visible), las páginas (no visibles) de resultados en forma tabular y la de la gráfica de la red. La página de entrada muestra la tabla de actividades



En la figura aparece la lista de 12 actividades con su vector de precedentes. La red se obtiene al oprimir el botón rotulado *Crear red* y los resultados se presentan en forma tabular y de manera gráfica.

Las siguientes figuras muestran las dos panatallas de salida: tabular y gráfica.

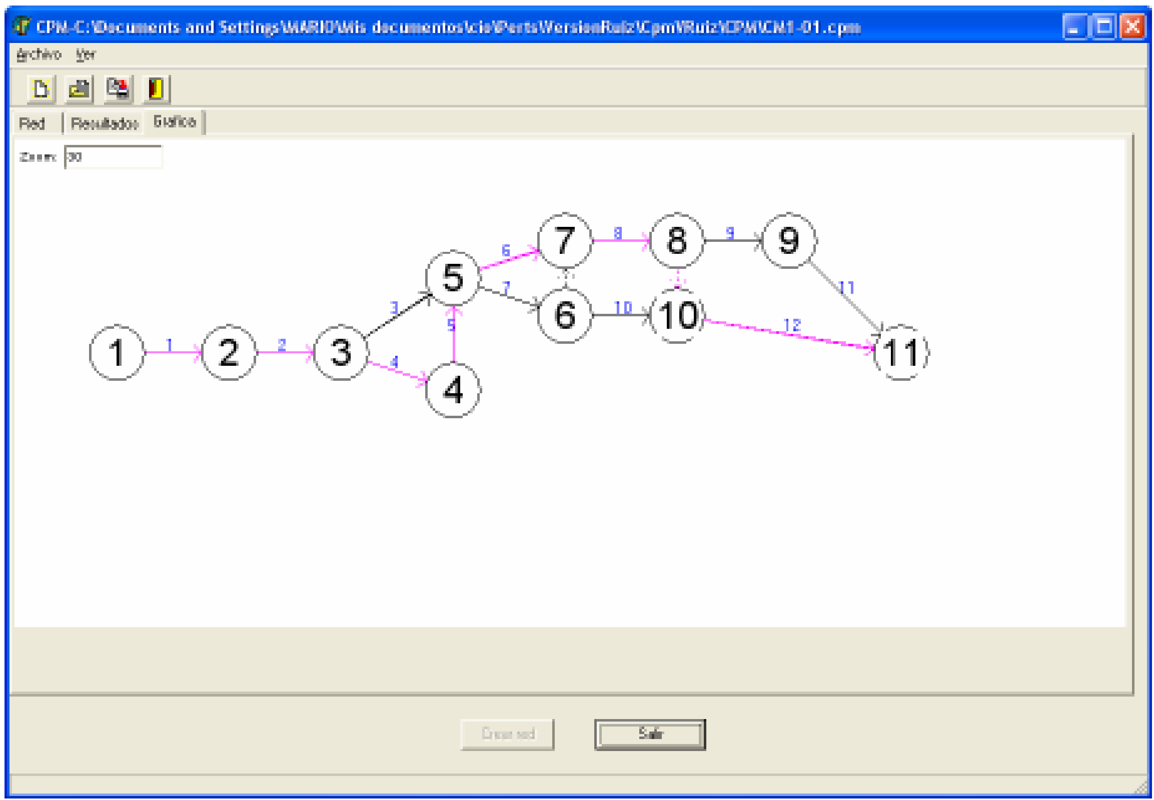


Número	i	j	Duración	I. Temprano	T. Temprano	I. Tardío	T. Tardío	H. Total	H.Libre	H. Interfcia.
1	1	2	2.0	0.0	2.0	0.0	2.0	0.0	0.0	0.0
2	6	7	0.0	19.0	19.0	22.0	22.0	3.0	3.0	0.0
3	8	10	0.0	28.0	28.0	28.0	28.0	0.0	0.0	0.0
4	2	3	3.0	2.0	5.0	2.0	5.0	0.0	0.0	0.0
5	3	5	5.0	5.0	10.0	11.0	16.0	6.0	6.0	0.0
6	3	4	7.0	5.0	12.0	5.0	12.0	0.0	0.0	0.0
7	4	5	4.0	12.0	16.0	12.0	16.0	0.0	0.0	0.0
8	5	7	6.0	16.0	22.0	16.0	22.0	0.0	0.0	0.0
9	5	6	3.0	16.0	19.0	19.0	22.0	3.0	0.0	3.0

Actividad	i	j
1	1	2
2	8	10
3	2	3
4	3	4

Actividad	i	j
1	6	7
2	8	10

Salida tabular



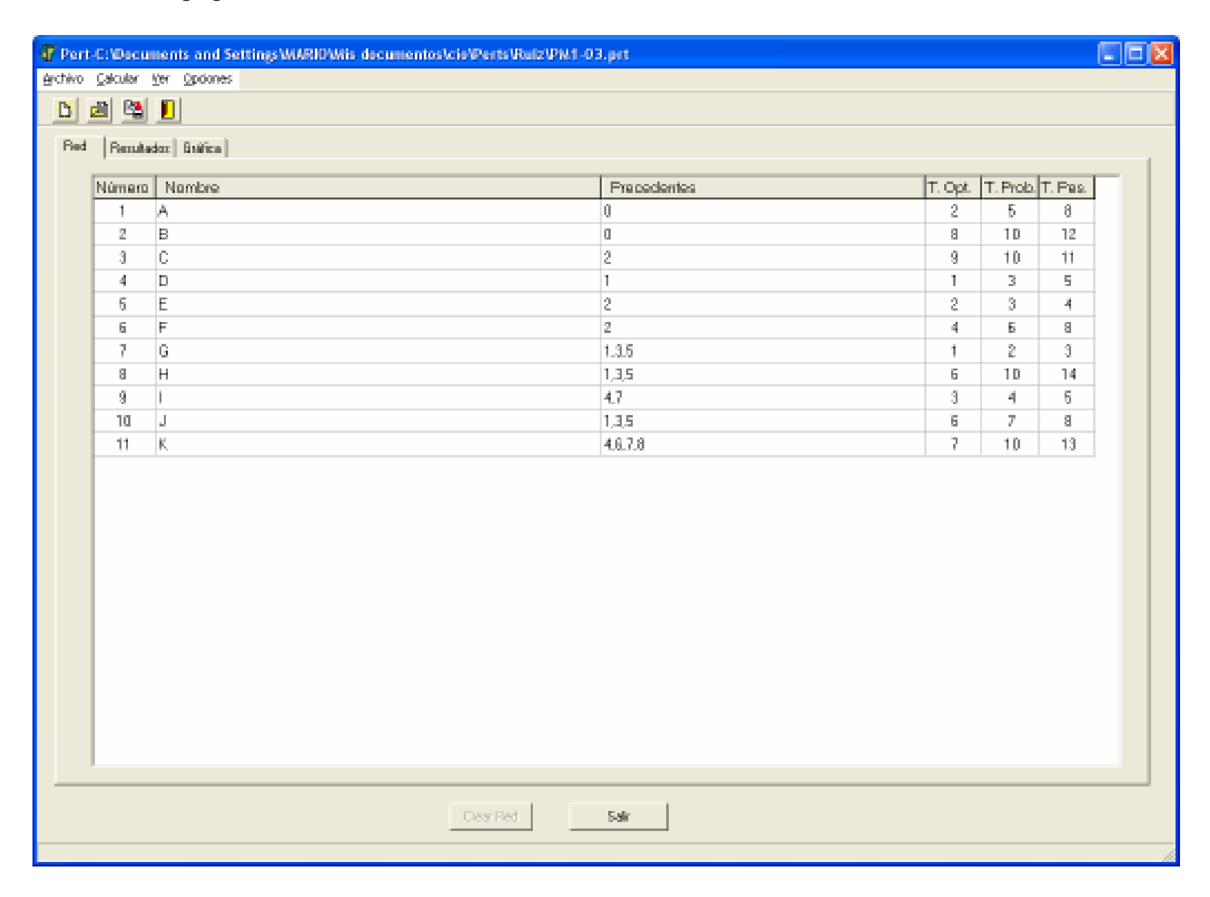
Salida gráfica

SysPert

Este programa crea una red a partir de la lista de actividades y sus precedentes utilizando el programa *SysRed*. El modelo utilizado se denomina AOA (actividades en los arcos). Los módulos que constituyen este programa son Principal, UMain, UDatos, Ucode, Ured, Ucpm, UValZ y UPert. El módulo Principal inicializa el proceso, crea el formulario y corre la aplicación.

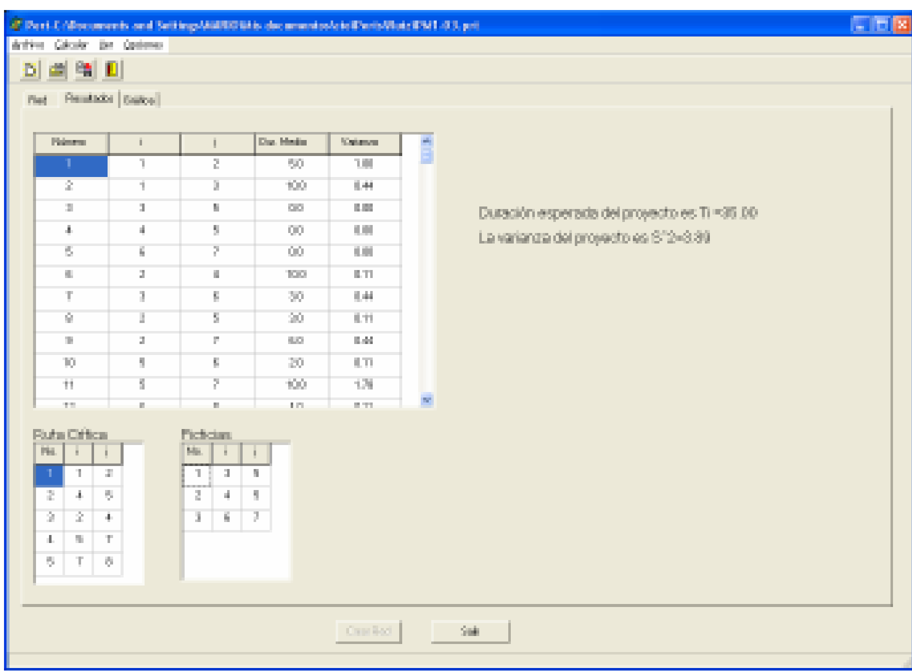
UMain es el módulo básico y opera prácticamente toda la aplicación. Contiene al formulario principal y es la interfaz gráfica del usuario; se utiliza tanto para la captura de datos como para el despliegue de resultados.

La siguiente figura muestra el formulario con el menú principal, barra de herramientas, la página de entrada de datos (visible), las páginas (no visibles) de resultados en forma tabular y la de la gráfica de la red. La página de entrada muestra la tabla de actividades



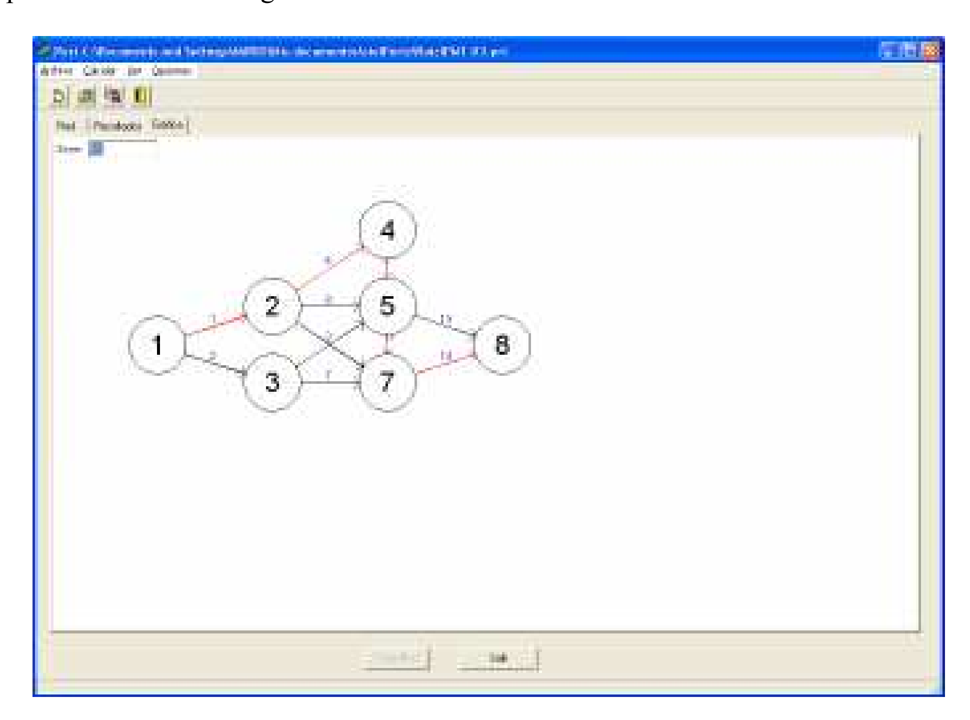
Número	Nombre	Precedentes	T. Opt.	T. Prob.	T. Pes.
1	A	0	2	5	8
2	B	0	8	10	12
3	C	2	9	10	11
4	D	1	1	3	5
5	E	2	2	3	4
6	F	2	4	6	8
7	G	1,3,5	1	2	3
8	H	1,3,5	6	10	14
9	I	4,7	3	4	5
10	J	1,3,5	6	7	8
11	K	4,6,7,8	7	10	13

Con estos datos al oprimir el botón *Crear Red* se construye la red de actividades y se calculan los pares *(i, j)* que corresponden al evento inicial *(i)* y al evento terminal *(j)* de cada actividad. También se calcula la duración esperada y la varianza. La segunda figura muestra la página con estos resultados, además se pueden observar la tabla de actividades que forman la ruta crítica y la tabla de actividades ficticias, así como la duración esperada del proyecto *Ti* y la varianza total.



Pantalla con las tablas de resultados mostrando los pares (i, j), duración y varianza esperadas.

En esta pantalla se muestra la gráfica de la red.



Los pares (i, j) determinados por el software, se muestran con los números correspondientes en los nodos de la red .

En la tabla resultados, el usuario puede utilizar la opción *Calcular* en menú principal para obtener:

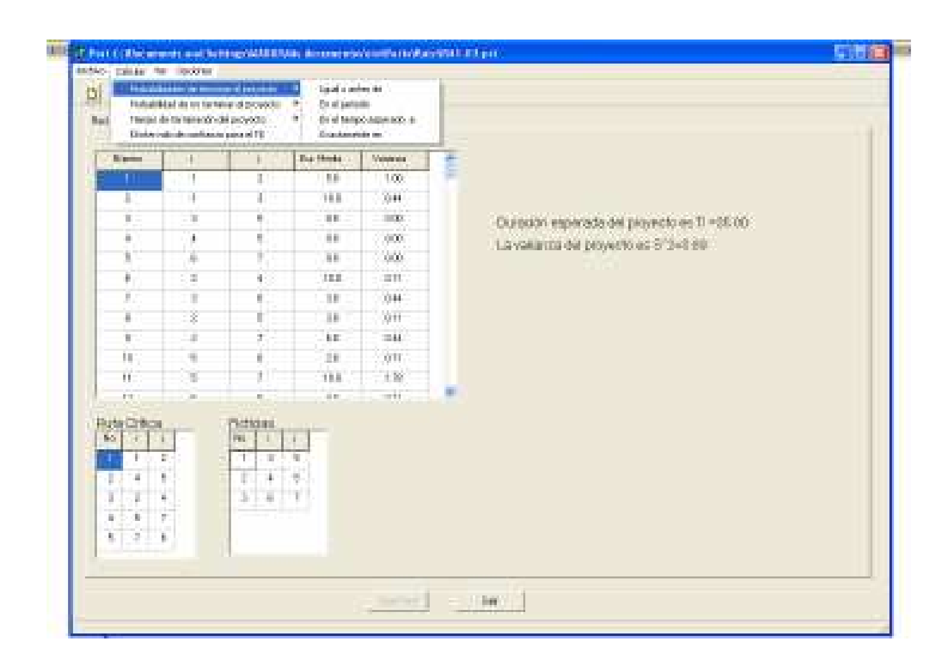
Las probabilidades de terminar el proyecto
- Igual o antes de…
- En el periodo…
- En el tiempo esperado ± *d*…
- Exactamente en…

Probabilidad de no terminar el proyecto
- En el tiempo programado TP o antes…
- Entre dos tiempos programados TP1 y TP2…

Tiempo de terminación del proyecto con probabilidad…

El intervalo de confianza para el tiempo esperado TE…

La siguiente figura muestra algunas de las opciones mencionadas.



El formulario opera 26 manejadores de eventos:

1. btAddActsClick : Añade actividades a la lista
2. FormActivate : Inicializa algunas variables antes de abrir el formulario
3. btAuxClick : Actualmente inhabilitado, se utilizó para pruebas
4. btGrabarClick : Graba en un archivo la lista de actividades actual
5. btRecuperarClick : Abre un archivo pregrabado con una lista de actividades, actualmente inhabilitado
6. Terminar1Click : Cierra la aplicación
7. CargarDatos2: Carga el vector VEstima con base en VActiva1 y los tiempos de la tabla de datos

8. btCreaRedClick : Inicia los procesos de: preparación de datos, creación de la red, cálculo de probabilidades y exhibición de resultados
9. btSalirClick : Cierra la aplicación
10. FormCreate : Crea el formulario, MiCodigos y Grafica
11. sgActiviDatosKeyPress: Crea una actividad vacía al oprimir <Enter>
12. Antesde1Click : Calcula la probabilidad de terminar el proyecto antes de su TP
13. Terminarelproyectoenelperiodo1Click : Calcula la probabilidad de terminar el proyecto en el periodo T1-T2
14. AlrededordesuTP1Click : Calcula la probabilidad de terminar el proyecto en su TE±d
15. Exactamenteen1Click : Calcula la probabilidad de terminar el proyecto exactamente en T
16. Antesde2Click : Calcula la probabilidad de no terminar el proyecto antes de su TP
17. Enelperiodo1Click : Calcula la probabilidad de no terminar el proyecto dentro de un determinado periodo T1-T2
18. Conprobabilidad1Click : Calcula el tiempo de terminación del proyecto con una probabilidad dada
19. Abrirundocumento1Click : Abre un archivo pregrabado con una lista de actividades
20. Guardarundocumento1Click : Graba en un archivo la lista de actividades actual
21. Precedentes1Click : Marca la opción ver precedentes
22. Ficticias1Click : Conmuta para hacer visibles o no las actividades ficticias
23. probOcurrEvTPClick : Calcula la probabilidad de que ocurra un determinado evento en un tiempo dado
24. FormClose: Cierra el formulario y destruye Grafica y MiCodigos
25. Vercomplemento1Click : Conmuta la opción verComplemento
26. tbtNuevoClick : Crea una nueva lista de actividades

Los procedimientos independientes del formulario son:

1. IniciaFormas: Prepara las tablas de datos y resultados con encabezados y ancho de columnas
2. Resetear: Reinicializa diversos objetos utilizados por la aplicación: vectores y colas
3. MiGrafica: Prepara los datos y crea los elementos para la gráfica de la red
4. CreaGrupos: Crea los grupos de actividades precedentes
5. CargaDatos: Carga el vector VActiva1 y la Cola1 con los valores de la tabla de datos
6. ReCargaDatos: Carga la tabla de resultados con los valores calculados
7. PrepararDatos: Prepara los datos para su proceso asignado códigos a las actividades

La unidad *UPert* funciona como enlace con otro de los módulos sustantivos del paquete, la unidad *UValz* donde se realizan todos los cálculos probabilísticos basados en la distribución normal. En *UMain* las ocho opciones del menú *Calcular* formulan una petición de cálculo, por ejemplo: *la probabilidad de terminar el proyecto antes de su tiempo programado TP*, ejecuta las siguientes instrucciones para el módulo UPert:

```
rParm[1]:='1';
rParm[2]:=FloatToStr((StrToFloat(sTP)-Mu)/Sigma);
rParm[3]:='1';
cPert(rParm);
```

y así se configura la línea de parámetros para la petición con el arreglo *rParm* de tres elementos:

rParm → ('1', FloatToStr((StrToFloat(sTP)-Mu)/Sigma), '1')

la cual será interpretada en UPert por la siguiente estructura *case* de opción múltiple:

```
case StrToInt(cParm[1]) of
  1:CalcAreaZ(StrToFloat(cParm[2]),StrToInt(cParm[3]));
  2:CalcAreaZZ(StrToFloat(cParm[2]),StrToFloat(cParm[3]),StrToInt(cParm[4]));
  3:CalcAreaZZ(StrToFloat(cParm[2]),StrToFloat(cParm[3]),StrToInt(cParm[4]));
  4:CalcAreaZZ(StrToFloat(cParm[2]),StrToFloat(cParm[3]),StrToInt(cParm[4]));
  5:CalZIzq(StrToFloat(cParm[2]),StrToFloat(cParm[3]),StrToFloat(cParm[4]),StrToInt(cParm[5]));
 end;{case}
```

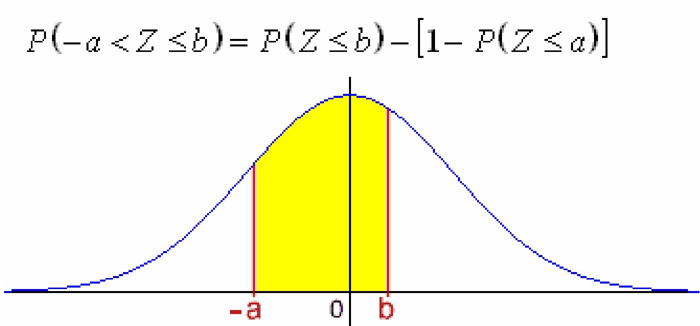
Cada una de las cinco opciones del *case* precedente invoca, con los parámetros correspondientes, alguno de los tres módulos: *CalcArearZ*, *CalcAreaZZ*, *CalZIzq*, de la unidad *UValz* los cuales se describen con amplitud más adelante.

UNIDAD UValz

UValz resuelve a través de métodos numéricos de integración, el problema del cálculo del área bajo la curva de la función que representa la distribución normal o Campana de Gauss y que permite obtener la probabilidad de que un evento ocurra en un determinado tiempo, es el caso del tiempo de terminación de una actividad o de toda la red de actividades. También resuelve el problema contrario: calcular el tiempo de ocurrencia de un evento en una red de actividades, dada una probabilidad de que suceda.

Mediante la fórmula trapecial calcularemos el valor de la integral: $\frac{1}{\sqrt{2\Pi}}\int_a^b e^{\frac{-x^2}{2}}dx$

Cuya gráfica es:



DISTRIBUCIÓN NORMAL

La función expresada en términos de Pascal es:

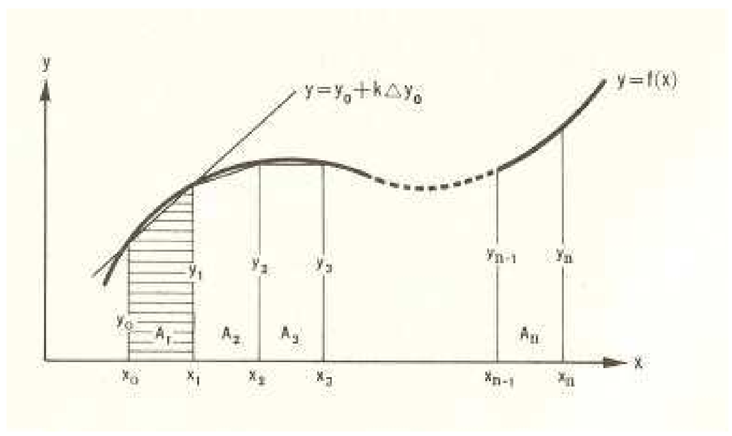
```
function Y(x:real):real;
begin
 Y:=Exp(-(x*x)/2)/Sqrt(2*pi);
end;{Y}
```

Para calcular el área bajo la curva se utilizó el método trapecial descrito por el Dr. Rodolfo Luthe García[1]

$$\int_{x_0}^{x_1} f(x)dx \; ; \; \frac{h}{2}(y_0 + y_1)$$

"Geométricamente, el primer miembro de la expresión anterior equivale al área A_1 bajo la curva $y=f(x)$ entre las rectas $x = x_0$, $x = x_1$ y el eje x. El segundo miembro, el valor aproximado de la integral representa el área del trapecio formado por las tres rectas antes mencionadas y la que une los puntos pivotes usados en la interpolación lineal, tal como se muestra en la siguiente figura."

[1] Rodolfo Luthe, Antonio Olivera y Fernando Schutz, Métodos Numéricos, (México D.F., México: Editorial Limusa 1978), pp 170-181.



Aproximación polinomial (lineal) del área bajo una curva

Traducido lo anterior al código Pascal en el paquete de software resulta la función que calcula el área bajo la curva *AreaBC*:

```
function AreaBC(A,B:real;N:integer):real;
//Calcula el área bajo la curva con la fórmula trapecial (Luthe pp. 170-181)
var
 i,j,M:integer;
begin
 for i:=1 to 2 do begin
  H:=(B-A)/N;
  X:=A;
  Area[i]:=0; //inicializa la i-ésima sumatoria
  M:=N-1; //número de subintervalos para el cálculo
  for j:=1 to M do begin //ciclo que obtiene la sumatoria de las áreas de los subintervalos
   X:=X+H;
   Area[i]:=Area[i]+Y(X);
  end;{for j}
  Area[i]:=H*(Y(A)+Y(B)+2*Area[i])/2; //fórmula trapecial
  N:=2*N;
 end;{for i}
 Rich:=(4*Area[2]-Area[1])/3; //interpolación de Richardson entre las dos sumatorias: N y 2N subintervalos
 AreaBC:=Rich;
end;{AreaBC}
```

La función *AreaBC* la utilizan los procedimientos : *CalcArearZ*, *CalcAreaZZ*, *CalZIzq*, y la función *Kalcula* de *UValz*.

La función *Kalcula* es una aportación relevante y original del grupo de investigación, resuelve el cálculo del valor de la variable aleatoria en la distribución normal a partir del valor del área bajo la curva. La técnica consiste en una exploración de izquierda a derecha en la curva utilizando un par de valores límite de un pequeño intervalo (b_1,b_2) hasta que el valor buscado (x) de la variable aleatoria quede entre ambos. Después se realiza una fase de precisión aproximando los valores exploradores b_1 y b_2 al valor buscado x para obtener los mejores resultados. Al respecto, se utilizan dos factores de aproximación, f_1 para el límite inferior b_1 y f_2 para el límite superior b_2 del intervalo explorador.

La función *Kalcula* implementa en Pascal la técnica descrita.

```
function Kalcula(AreaTar:real):real;
//Realiza la exploración para determinar la Z buscada a partir de un área determinada
var
 H:real;
 Izq:boolean;
 f1,f2, //factores de aproximación en la exploración
 ArZ:real;
 b1,b2, //límites del intervalo de exploración
 Area1,Area2:real;
 ZNeg:boolean;
 k:integer;
begin
   K1:=0;
   K2:=0;
   k:=0;{contador}
   ZNeg:=False;
   A:=-4;  {Valor inicial para la variable aleatoria en el límite inferior del rango explorador}
   B:=-3.9;  {Valor inicial para la variable aleatoria en el límite superior del rango
explorador}
   N:=16;   {Número de intervalos en dividirá el rango explorador}
   H:=(B-A)/N; {ancho del intervalo para el rango explorador}
   if AreaTar=0.5 then Area1:=AreaTar  {variable aleatoria en el centro de la distribución}
   else begin
    if AreaTar<0.5
     then begin   {utilizado para realizar los cálculos en el lado positivo simétrico de la curva}
      AreaTar:=1-AreaTar;
      ZNeg:=True;
     end;
    b2:=B; {b2:=b1; {valor inicial del límite superior del primer intervalo}
    Area2:=AreaBC(A,b2,N); {área del límite superior del primer intervalo}
    repeat  {cálculo de áreas sucesivas hasta revasar el área de la variable aleatoria desconocida}
     inc(K1);
     inc(k);{espía}
     b1:=b2;
     Area1:=Area2; {preserva el área de la cota inferior del intervalo}
     b2:=b2+H;  {avanza al siguiente intervalo}
     Area2:=AreaBC(A,b2,N); {calcula el área del nuevo limite superior del intervalo}
    until (Area2>=AreaTar);//Se detiene el proceso cuando el intervalo envuelve a la variable aleatoria buscada
    f1:=(AreaTar-Area1)/AreaTar; {primer factor de ajuste para acercar el límite b1 a bx}
    K1:=k;
    f2:=(Area2-AreaTar)/AreaTar; {segundo factor de ajuste para acercar el límite b2 a bx}
    while b1<b2 do begin  {ciclo utilizado para cruzar los valores exploradores b1 y b2}
     inc(K2);
     if b1>=0
       then b1:=b1*(1+f1)
       else b1:=b1*(1-f1);
     if b2>=0
      then b2:=b2*(1-f2)
```

```
      else b2:=b1*(1+f2);
    end;{while}
    b1:=(b1+b2)/2;
    Area1:=AreaBC(A,b1,N);
   end;{else del if AreaTar=0.5}
  if ZNeg then b1:=-b1; {para valores simétricos}
 Kalcula:=b1; {valor buscado}
end;
```

UValz debe su nombre precisamente a este cálculo de la variable aleatoria a partir del área conocida bajo la curva. Las tablas ordinarias de Z solamente se pueden utilizar para el cálculo inverso, es decir para obtener el área bajo la curva a partir del valor de la variable aleatoria.

UNIDAD UDatos

UDatos contiene las estructuras de datos definidas para el paquete completo de software. En este módulo se definen todos los tipos de datos necesarios como las clases, arreglos lineales y tablas, pilas, colas, registros, etc. También se declaran las variables globales.

Clases:
WVector
TVEstima
StatusType
StackNode
Snack
QueueNode
AVector
TVcpm
GpoVector
TFicVector

Tipos:
TRIndex
TEstima
OneInt
OneIntPtr
OneEstima
OneEstimaPtr
StackNodePtr
QueueNodePtr
VisitProc
TActiva
TVActiva
TRVcpm
TRFic
TRC
TRLin6
TRLin8
TRLin9
TStr4
TcParm
TVP
TGrupo
TRGpos
OneActiva
OneActivaPtr
OneCPM
OneCPMPtr
OneGpo
OneGpoPtr
OneFic
OneFicPtr

Constantes

maxGpos

Variables y su tipo asociado:
Gpo,VDura:GpoVector;
JGpo:WVector;
VActiva1:AVector;
Vcpm:TVcpm;
FicVector:TFicVector;
VEstima:TVEstima;
qHol,
ColaFic,
Cola1,
cGpos:Queue;
RIndex:TRIndex;
JMax,nT:integer;
DurPro:real;{integer;}
cc32,
flEncolar,
fliP,
flJMax:boolean;
sTP,NomAr:string;
sigm,
Mu,Sigma:real;

En el apéndice anexo al informe impreso se presenta un diccionario de datos con referencias cruzadas entre todas las unidades, conteniendo todas las variables empleadas en el paquete.

APENDICE

(LISTADOS DEL CÓDIGO DE TODOS LOS MÓDULOS)

```
unit UMain;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Grids, StdCtrls, Menus, ComCtrls, ActnList, ImgList, ToolWin, miunit, UGraficaNodos,
  ExtCtrls;

type
  TfmMain = class(TForm)
    btAddActs: TButton;
    btCreaRed: TButton;
    btGrabar: TButton;
    btRecuperar: TButton;
    btSalir: TButton;
    MainMenu1: TMainMenu;
    Archivo1: TMenuItem;
    Nuevo1: TMenuItem;
    Terminar1: TMenuItem;
    StatusBar1: TStatusBar;
    Abrirunared1: TMenuItem;
    ActionList1: TActionList;
    ImageList1: TImageList;
    Calcular1: TMenuItem;
    CPM3: TMenuItem;
    PERT3: TMenuItem;
    ToolBar1: TToolBar;
    tbtNuevo: TToolButton;
    ToolButton2: TToolButton;
    ToolButton3: TToolButton;
    ToolButton4: TToolButton;
    ToolButton5: TToolButton;
    Ver1: TMenuItem;
    Ficticias1: TMenuItem;
    N2: TMenuItem;
    Precedentes1: TMenuItem;
    PageControl1: TPageControl;
    TabSheet1: TTabSheet;
    TabSheet2: TTabSheet;
    sgActiviDatos: TStringGrid;
    sgFic: TStringGrid;
    sgCPM: TStringGrid;
    sgRC: TStringGrid;
    lbRC: TLabel;
    lbFicticias: TLabel;
    Label2: TLabel;
```

```
Antesde1: TMenuItem;
Terminarelproyectoenelperiodo1: TMenuItem;
AlrededordesuTP1: TMenuItem;
Exactamenteen1: TMenuItem;
Antesde2: TMenuItem;
Enelperiodo1: TMenuItem;
Tiempodeterminacindelproyecto1: TMenuItem;
Conprobabilidad1: TMenuItem;
ElintervalodeconfianzaparaelTE1: TMenuItem;
Opciones1: TMenuItem;
lbProba: TLabel;
lbProbaCom: TLabel;
Action1: TAction;
OpenDialog1: TOpenDialog;
SaveDialog1: TSaveDialog;
Abrirundocumento1: TMenuItem;
Guardarundocumento1: TMenuItem;
ToolButton8: TToolButton;
ToolButton9: TToolButton;
TabSheet3: TTabSheet;
PnGraficar: TPanel;
Vercomplemento1: TMenuItem;
Label1: TLabel;
lbTituloPERT: TLabel;
procedure btAddActsClick(Sender: TObject);
procedure FormActivate(Sender: TObject);
procedure btAuxClick(Sender: TObject);
procedure btCreaRedClick(Sender: TObject);
procedure btGrabarClick(Sender: TObject);
procedure btRecuperarClick(Sender: TObject);
procedure Terminar1Click(Sender: TObject);
procedure btSalirClick(Sender: TObject);
procedure FormCreate(Sender: TObject);
procedure sgActiviDatosKeyPress(Sender: TObject; var Key: Char);
procedure Antesde1Click(Sender: TObject);
procedure Terminarelproyectoenelperiodo1Click(Sender: TObject);
procedure AlrededordesuTP1Click(Sender: TObject);
procedure Exactamenteen1Click(Sender: TObject);
procedure Antesde2Click(Sender: TObject);
procedure Enelperiodo1Click(Sender: TObject);
procedure Conprobabilidad1Click(Sender: TObject);
procedure Abrirundocumento1Click(Sender: TObject);
procedure Guardarundocumento1Click(Sender: TObject);
procedure Precedentes1Click(Sender: TObject);
procedure Ficticias1Click(Sender: TObject);
procedure probOcurrEvTPClick(Sender: TObject);
procedure CargarDatos2;
```

```
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure Vercomplemento1Click(Sender: TObject);
    procedure tbtNuevoClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  fmMain: TfmMain;
  Grafica:Aco;

implementation

uses UDatos,URed, UCPM, Upert;

{$R *.DFM}

procedure IniciaFormas;
//Prepara las tablas de datos y resultados con encabezados y ancho de columnas
begin
 with fmMain do begin
  with sgActiviDatos do begin //inicializa el StringGrid
  Font.Size:=11;
  ColWidths[0]:=55;
  ColWidths[1]:=400;
  ColWidths[2]:=290;
  ColWidths[3]:=50;
  ColWidths[4]:=50;
  ColWidths[5]:=50;
  Cells[0,0]:='Número';
  Cells[1,0]:='  Nombre';
  Cells[2,0]:='  Precedentes';
  Cells[3,0]:='T. Opt.';
  Cells[4,0]:='T. Prob.';
  Cells[5,0]:='T. Pes.';
  Cells[0,1]:=Format('%8d',[1]);
  Width:=GridWidth+25;
  Left:=fmMain.TabSheet1.Left+(fmMain.TabSheet1.Width-Width)div 2;//centra
StringGrid en TabSheet1
 end;{with sgActiviDatos}
 sgFic.Cells[0,0]:=' No.';
 sgFic.Cells[1,0]:='    i';
 sgFic.Cells[2,0]:='    j';
 sgFic.Visible:=False;
 lbFicticias.Visible:=False;
```

```
 sgCPM.Cells[0,0]:='      Número';
 sgCPM.Cells[1,0]:='           i';
 sgCPM.Cells[2,0]:='           j';
 sgCPM.Cells[3,0]:=' Dur. Media';
 sgCPM.Cells[4,0]:='    Varianza';{'         iPx';}
 sgCPM.Cells[5,0]:='         tPx';
 sgCPM.Cells[6,0]:='         iLx';
 sgCPM.Cells[7,0]:='         tLx';
 sgCPM.Cells[8,0]:='        hTotal';
 sgCPM.Cells[9,0]:='        hLib';
 sgCPM.Cells[10,0]:='      hInter';
 btCreaRed.Enabled:=False;
 lbRC.Visible:=False;
 lbTituloPERT.Visible:=False;
 Label2.Visible:=False;
 Label1.Visible:=False;
 lbProba.Visible:=False;
 lbProbaCom.Visible:=False;
 TabSheet2.Enabled:=True;
 TabSheet2.Visible:=True;
 PageControl1.ActivePage:=TabSheet1;
 sgCPM.Visible:=False;
 sgRC.Visible:=False;
 sgActiviDatos.SetFocus;
 sgFic.Font.Size :=8;
 sgCPM.Font.Size :=8;
 Label2.Font.Size :=12;
 LbRc.Font.Size :=12;
 LbFicticias.Font.Size :=12;
 Grafica.hecha:=false;
end;{with fmMain}
end;{IniciaFormas}

procedure Resetear;
//Reinicializa diversos objetos utilizados por la aplicación: vectores y colas
begin
 Cola1.Clear;
 VActiva1.Done;
 {Gpo.Done; }
 VDura.Done;
 JGpo.Done;
 VCpm.Done;
 FicVector.Done;
 VEstima.Done;
end;

procedure MiGrafica;
```

```
//Envía las (i,j) de las actividades para graficar la red
Var
I,K:Integer;
Tipos: array of word;
L:Word;
begin
setLength(Tipos,FmMain.SgCPM.RowCount-1 );


Grafica.DarTamano(FmMain.SgCPM.RowCount-1 );
 For I:=0 to FmMain.SgCPM.RowCount-2  do  //para guardar las I J de las actividades
    begin
    if ((AnsiPos(' 0.0', FmMain.SgCPM.cells[3,I+1]))=0) then
    begin
    Tipos[I]:=0;
    end
    else
    Tipos[I]:=2;
    end;

   For k:=0 to High(Indicedos)  do  //para guardar las I J de las actividades
    begin
    if (Tipos[Indicedos[k]-1]=0) then
        Tipos[Indicedos[k]-1]:=1
         else Tipos[Indicedos[k]-1]:=3;
    end;       //ya encontre los tipos de actividades

    k:=1;

   for I:=0 To High(Tipos) do
    begin
    if (Tipos[I]<=1) then
    begin
    L:= StrToInt(FmMain.SgCPM.cells[1,I+1]);
     Grafica.GuardarCordenadas (K,L, StrToInt(FmMain.SgCPM.cells[2,I+1]) ,Tipos[I]);
     inc(k);
    end;
    end;

    for I:=0 To High(Tipos) do
    begin
    if (Tipos[I]>1) then
    begin
     Grafica.GuardarCordenadas                    (K,StrToInt(FmMain.SgCPM.cells[1,I+1]),
StrToInt(FmMain.SgCPM.cells[2,I+1]) ,Tipos[I]);
     inc(k);
    end;
```

```
   end;
end;{MiGrafica}

procedure CreaGrupos;
//Crea los grupos de actividades precedentes
var
 A1,Ap,nlAp:TVActiva;
 GpoVP:GpoVector;
 aPiv,aVP:word;
 k,gk,pk:word;
 mLVP,nFin:integer; //mLVP=mínimo LVP,
 cVP32,cVPG:Queue;
 flg,fl32:boolean;
 gS,gSS:ShortString;
 sVP:ShortString;
begin {CreaGrupos}
 cVP32.Init(SizeOf(word),maxGpos);//inicializa cola de actividades con cM32, únicamente
Index's
 cVPG.Init(SizeOf(word),maxGpos); //inicializa cola con VP's para el vector de grupos

 for k:=1 to VActiva1.VSize do {crea cola de actividades con VP cM3=2}
  begin
   flG:=False;
   VActiva1.Recall(k,A1);
   sVP:='';
   pk:=1;
   if A1.LVP>1 then //selecciona actividades con cM32
    repeat
     if A1.VP[pk]<>',' //brinca las comas y convierte a número el elemento del VP
      then sVP:=sVP+A1.VP[pk]
      else begin
       VActiva1.Recall(StrToInt(sVP),Ap);
       flG:=Ap.cM3=2; //verifica si el elemento del VP tiene cM32
       sVP:='';
      end;
     inc(pk);
    until flG or (pk>Length(A1.VP));
    if (A1.LVP>1)and not flG {necesario para incluir en el análisis a la última actividad de
A1.VP}
     then begin
      VActiva1.Recall(StrToInt(sVP),Ap);
      flG:=Ap.cM3=2;
     end;
   if flG then cVP32.EnQueue(A1.Index,SizeOf(A1.Index));{encola actividades Ap con
cM32}
  end;{for k:=1....crea cola de actividades con VP cM3=2}
```

```
 if cVP32.Items_In>0 then begin   {Filtro1:elimina actividades que no empiecen Ap con
cM32}
 sVP:='';{necesaria para el Filtro1}
 for k:=1 to cVP32.Items_In do
  begin
   cVP32.DeQueue(k,SizeOf(k));
   VActiva1.Recall(k,A1);
   pk:=1;
   if A1.LVP>1 then //brinca las comas y convierte a número el elemento del VP
    repeat
     if A1.VP[pk]<>','
      then sVP:=sVP+A1.VP[pk]
      else begin
       VActiva1.Recall(StrToInt(sVP),Ap);
       flG:=Ap.cM3=2;
       if flG
        then cVP32.EnQueue(k,SizeOf(k));{reencola las que tienen cM32, las demás las
desecha}
       sVP:='';
      end;
     inc(pk);
    until sVP='';
  end;{for k:=1 to cVP32.Items_In}

 repeat {Filtro2:elimina actividades con VP repetido}
  cVP32.DeQueue(aPiv,SizeOf(aPiv));//decola pivote
  cVPG.EnQueue(aPiv,SizeOf(aPiv)); //guarda pivote en cola para el vector de grupos
  VActiva1.Recall(aPiv,A1);
  cVP32.DeQueue(aVP,SizeOf(aVP)); //decola la siguiente acividad con cM32
  if cVP32.Last_Op_Ok //verifica que la cVP32 esté correcta
   then begin
    VActiva1.Recall(aVP,Ap);
    if A1.VP<>Ap.VP //verifica VP repetidos
     then cVP32.EnQueue(aVP,SizeOf(aVP));//reencola las no repetidas
   end;{if cVP32.Last_Op_Ok}
 until cVP32.Is_Empty;

 if cVPG.Items_In>0 {crea los grupos}
  then begin
   GpoVP.Init(cVPG.Items_In);  //inicializa vector de grupos
   for k:=1 to cVPG.Items_In do {de la cola cVPG carga vector de las VP's con cM32}
    begin
     cVPG.DeQueue(aVP,SizeOf(aVP)); //carrusel en cVPG
     cVPG.EnQueue(aVP,SizeOf(aVP));
     VActiva1.Recall(aVP,Ap);
     GpoVP.Store(k,IntToStr(Ap.LVP)+','+Ap.VP);{carga vector de las VP's con cM32,
formato:(LVP,VP)}
```

```
    end;{for k:=1 to cVPG.Items_In}

   {Calcula mVLP: mínimo LVP}
   gk:=0;{inicializa el índice de grupos}
   nFin:=0;
   repeat
    mLVP:=MaxInt;
    for k:=1 to GpoVP.VSize do begin
     GpoVP.Recall(k,gS);
     if (StrToInt(gS[1])>0)and(mLVP>StrToInt(gS[1]))
      then mLVP:=StrToInt(gS[1]); //captura el mínimo LVP en el vector de grupos GpoVP
    end;{for k:=1 to GpoVP.VSize}

   {Forma grupo con mLVP}
    flG:=False;
    k:=0;
    repeat
     inc(k);
     GpoVP.Recall(k,gS);
     if StrToInt(gS[1])=mLVP
      then begin
       inc(gk);
       gSS:=Copy(gS,Pos(',',gS)+1,Length(gS)-Pos(',',gS));//copia en gSS el VP
       if gSS<>',' then Gpo.Store(gk,gSS);
       gS[1]:='0';
       GpoVP.Store(k,gS);
       flG:=True;
       inc(nFin);
      end;
    until flG or (k>=GpoVP.VSize);

    {3.- Suprimir el grupo formado de todos los VP's que lo contengan}
    for k:=1 to GpoVP.VSize do begin
     GpoVP.Recall(k,gS);
     Gpo.Recall(gk,sVP);
     if (gS[1]<>'0')and(Pos(sVP,gS)>0)and(Gpo.VectorOpStatus)
      then begin
       Delete(gS,Pos(sVP,gS),Length(sVP));
       GpoVP.Store(k,gS);
      end;
    end;{for:=1 to GpoVP.VSize}
   until not(Gpo.VectorOpStatus)or(nFin>=GpoVP.VSize); {en M3-02 nunca se hace falsa,
modificar condición}
  end;{if cVPG.Items_In>0}
 end;{if cVP32.Items_In>0}
end;{CreaGrupos}
```

```
procedure CargaDatos;
//Carga el vector VActiva1 y la Cola1 con los valores de la tabla de datos
var
 k:word;
 rActiva:TVActiva;
begin
 with fmMain.sgActiviDatos do
 begin
  VActiva1.Init(RowCount-1);
  VEstima.Init(RowCount-1);
  VDura.Init(RowCount-1);
  nT:=0;
  for k:=1 to RowCount-1 do
   begin
    rActiva.Index:=StrToInt(Rows[k].Strings[0]);
    rActiva.Nombre:=Rows[k].Strings[1];
    rActiva.LVP:=0; {StrToInt(Rows[k].Strings[2]);}
    rActiva.VP:=Rows[k].Strings[2];
    rActiva.i:=0; {StrToInt(Rows[k].Strings[4]);}
    rActiva.j:=0; {StrToInt(Rows[k].Strings[5]);}
    rActiva.cApar:=0; {StrToInt(Rows[k].Strings[6]);}
    rActiva.cM3:=0; {StrToInt(Rows[k].Strings[7]);}
    rActiva.cR:=0; {StrToInt(Rows[k].Strings[8]);}
    {VDura.Store(k,fmMain.sgActiviDatos.Rows[k].Strings[9]);}
    rActiva.Ok:=0;
    rActiva.index2:=k;
    VActiva1.Store(k,rActiva);
    //if (rActiva.LVP>0)and(rActiva.cApar=0) then inc(nT);//**MA=considerar la variable
nT en sus rutinas
    Cola1.EnQueue(rActiva.Index,SizeOf(word));
   end;
 end;{with fmMain.sgActiviDatos}
 fmMain.btCreaRed.Enabled:=True;
end;{CargaDatos}

procedure ReCargaDatos;
//Carga la tabla de resultados con los valores calculados
var
 k,nk:integer;
 rActiva:TVActiva;
 rFic:TRFic;
 rRC:TRC;
 rEstima:TEstima;
 rCpm:TRVcpm;
 sDura:ShortString;
 X:real;
begin
```

```
fmMain.sgCPM.Visible:=True;
fmMain.sgRC.Visible:=True;
with fmMain.sgFic do
begin
 RowCount:=1;
 k:=0;
 if FicVector.VSize>0 {not(ColaFic.Is_Empty)}
  then begin
   Visible:=True;
   fmMain.lbFicticias.Visible:=True;
   repeat
    RowCount:=RowCount+1;
    FixedRows:=1;
    inc(k);
    FicVector.Recall(k,rFic);
    Rows[k].Strings[0]:=Format('%5d',[k]);{IntToStr(k);}
    Rows[k].Strings[1]:=Format('%5d',[rFic.i]);{IntToStr(rFic.i);}
    Rows[k].Strings[2]:=Format('%5d',[rFic.j]);{IntToStr(rFic.j);}
   until k=FicVector.VSize;
 end;
end;{with fmMain.sgFic}
with fmMain.sgRC do
begin
 RowCount:=1;
 k:=0;
 Visible:=True;
 fmMain.lbRC.Visible:=True;
 Cells[0,0]:='  No.';
 Cells[1,0]:='    i';
 Cells[2,0]:='    j';
 for nk:=1 to qHol.Items_In do begin {repeat}
  RowCount:=RowCount+1;
  FixedRows:=1;
  inc(k);
  qHol.DeQueue(rRC,SizeOf(rRC));
  qHol.EnQueue(rRC,SizeOf(rRC));{preserva qHol}
  Rows[k].Strings[0]:=Format('%5d',[k]);{IntToStr(k);}
  Rows[k].Strings[1]:=Format('%5d',[rRC.i]);{IntToStr(rRC.i);}
  Rows[k].Strings[2]:=Format('%5d',[rRC.j]);{IntToStr(rRC.j);}
 end;{for nk:=1 to qHol.Items_In} {until qHol.Is_Empty;}
end;{with fmMain.sgRC}
with fmMain.sgCPM do
begin
 RowCount:=1;
 k:=0;
 nk:=0;
 X:=0;
```

```
  repeat
   RowCount:=RowCount+1;
   FixedRows:=1;
   inc(k);
   Vcpm.Recall(k,rCpm);
   {VEstima.Recall(k,rEstima);}
   Rows[k].Strings[0]:=Format('%12d',[rCpm.Index]);{IntToStr(rCpm.Index);}
   Rows[k].Strings[1]:=Format('%12d',[rCpm.i]);
   Rows[k].Strings[2]:=Format('%12d',[rCpm.j]);{IntToStr(rCpm.j);}
   Rows[k].Strings[3]:=Format('%15.1f',[rCpm.Dur]);{FloatTostr(rCpm.Dur);}
   if rCpm.Dur=0
    then begin
     inc(nk);
     Rows[k].Strings[4]:=Format('%13.2f',[X]);
    end
    else begin
     VEstima.Recall(k-nk,rEstima);
     Rows[k].Strings[4]:=Format('%13.2f',[rEstima.Varianza]);
    end;
   {Rows[k].Strings[4]:=Format('%13.2f',[rEstima.Varianza]); {FloatTostr(rCpm.iPx);
   {Rows[k].Strings[4]:=Format('%10.2f',[rCpm.iPx]); {FloatTostr(rCpm.iPx);
   Rows[k].Strings[5]:=Format('%10.2f',[rCpm.tPx]);{FloatTostr(rCpm.tPx);
   Rows[k].Strings[6]:=Format('%10.2f',[rCpm.iLx]);{FloatTostr(rCpm.iLx);
   Rows[k].Strings[7]:=Format('%10.2f',[rCpm.tLx]);{FloatTostr(rCpm.tLx);
   Rows[k].Strings[8]:=Format('%10.2f',[rCpm.hTotal]);{FloatTostr(rCpm.hTotal);
   Rows[k].Strings[9]:=Format('%10.2f',[rCpm.hLib]);{FloatTostr(rCpm.hLib);
   Rows[k].Strings[10]:=Format('%10.2f',[rCpm.hInter]);{FloatTostr(rCpm.hInter);}
  until k=Vcpm.VSize;
 end;{with fmMain.sgFic}
 fmMain.label2.Caption:='Duración esperada del proyecto es Ti
='+Format('%f',[DurPro]);{FloatTostr(DurPro);}
 fmMain.label1.Caption:='La varianza del proyecto es S^2='+Format('%f',[Sigm]);
 fmMain.label1.Visible:=True;
 fmMain.label2.Visible:=True;
end;{ReCargaDatos}

procedure TfmMain.btAddActsClick(Sender: TObject);
//Añade actividades a la lista
begin
 {sgActiviDatos.Cells[0,sgActiviDatos.RowCount]:=IntToStr(sgActiviDatos.RowCount);
 sgActiviDatos.Cells[1,sgActiviDatos.RowCount]:=' A';
 sgActiviDatos.Cells[4,sgActiviDatos.RowCount]:=IntToStr(0);
 sgActiviDatos.Cells[5,sgActiviDatos.RowCount]:=IntToStr(0);
 sgActiviDatos.Cells[7,sgActiviDatos.RowCount]:=IntToStr(0);
 sgActiviDatos.RowCount:=sgActiviDatos.RowCount+1;
 sgActiviDatos.FixedRows:=1;
 btCreaRed.Enabled:=True;}
```

```
end;

procedure TfmMain.FormActivate(Sender: TObject);
//Inicializa algunas variables antes de abrir el formulario
begin
 Cola1.Init(SizeOf(TActiva),65535);
 Gpo.Init(maxGpos);
 IniciaFormas;
 Caption:='Pert-';
 StatusBar1.SimpleText:='Pulse <Intro> para introducir otra actividad';
 PageControl1.ActivePage:=TabSheet2;
 {sgActiviDatos.Visible:=True;}
 PageControl1.ActivePage:=TabSheet1;
 {sgActiviDatos.SetFocus;}
end;

procedure TfmMain.btAuxClick(Sender: TObject);
//Actualmente inhabilitado, se utilizó para pruebas
begin
 label2.Caption:=sgActiviDatos.Rows[sgActiviDatos.Row].Strings[0];
end;



procedure TfmMain.btGrabarClick(Sender: TObject);
//Graba en un archivo la lista de actividades actual
var
 S:string;
 f:file of TRLin6;
 k,kl:word;
 Lin:TRLin6;
begin
 S:=InputBox('Grabar archivo', 'Nobre del archivo', '')+'.prt';
 AssignFile(f,S);
 Rewrite(f);
 with fmMain.sgActiviDatos do
  for k:=1 to RowCount-1 do
   begin
    for kl:=0 to 6 do
     Lin[kl]:=Rows[k].Strings[kl];
    Write(f,Lin);
   end;
end;


procedure TfmMain.btRecuperarClick(Sender: TObject);
//Abre un archivo pregrabado con una lista de actividades, actualmente inhabilitado
```

```
var
 S:string;
 f:file of TRLin6;
 k,kl:word;
 Lin:TRLin6;
 Ok:boolean;
begin
 PageControl1.ActivePage:=TabSheet1;
 sgCPM.Visible:=False;
 sgRC.Visible:=False;
 {sgFic.Visible:=False;}
 Ok:=InputQuery('Abrir archivo', 'Nombre del archivo', NomAr);
 if Ok
 then begin
 NomAr:=NomAr+'.prt';
 if NomAr=''
  then btCreaRed.Enabled:=False
  else begin
   AssignFile(f,NomAr);
   Reset(f);
   fmMain.Caption:=NomAr;
   with fmMain.sgActiviDatos do
    begin
     k:=1;
     RowCount:=1;
     while not EOF(f) do
      begin
       Read(f,Lin);
       RowCount:=RowCount+1;
       FixedRows:=1;
       for kl:=0 to 6 do
        Rows[k].Strings[kl]:=Lin[kl];
        inc(k);
     end;{while}
    end;{with}
   btCreaRed.Enabled:=True;
   Calcular1.Enabled:=False;
  end; {if NomAr='' else}
 end;{if Ok then}
end;{btRecuperarClick}

procedure TfmMain.Terminar1Click(Sender: TObject);
//Cierra la aplicación
begin
 Application.Terminate;
end;
```

```
procedure preparardatos;
//Prepara los datos para su proceso asignando códigos a las actividades
begin
MiCodigos:=Tcodigos.Create;
with MiCodigos do
begin
 VerifComas;
 GuardarDatos;  //VERIFICA QUE LOS ELEMENTOS NO SOBREPASE LOS LIMITES
Y LOS GUARDA //REGRESA 0 SI ESTA BIEN 1 SI ESTA MAL
 CodAparicion;          //ME GENERA LA TABLA Y ME DICE SI VIENE SOLA O
ACOMPAÑADA
 Cm32;
 GuardaCM32;
 MiOrden;
 CrearMiDondeEstan;
 OrdenVP;
 CR;
 borrar;
end; //with MiCodigos
end; //preparardatos

{********************************}
procedure TfmMain.CargarDatos2;
//Carga el vector VEstima con base en VActiva1 y los tiempos de la tabla de datos
Var
k:Integer;
rEstima:TEstima;
rActiva:TVActiva;
//S:string;
begin
  for k:=1 to VEstima.MaxVectorSize do
   begin
    VActiva1.Recall(K,rActiva);
    rEstima.Index:=k;
    //S:=sgActiviDatos.Cells[3,rActiva.Index2];
    rEstima.tOp:=StrToFloat(sgActiviDatos.Cells[3,rActiva.Index2]);{????Aquí truena....}
    rEstima.tMe:=StrToFloat(sgActiviDatos.Cells[4,rActiva.Index2]);
    rEstima.tPe:=StrToFloat(sgActiviDatos.Cells[5,rActiva.Index2]);

rEstima.DurMe:=0.1666666666666666666666666666667*(rEstima.tOp+4*rEstima.tMe+
rEstima.tPe);
    rEstima.Varianza:=Sqr(0.1666666666666666666666666666667*(rEstima.tOp-
rEstima.tPe));
    VEstima.Store(rActiva.Index2,rEstima);
   end;
end;
{********************************}
```

```
procedure TfmMain.btCreaRedClick(Sender: TObject);
//Inicia los procesos de: preparación de datos, creación de la red, cálculo tiempos y
holguras,
//calculo de probabilidades y exhibición de resultados
begin
 CargaDatos;
 Preparardatos;
 CargarDatos2;
 {CreaGrupos;}
 ArmaRed;
 Cpm;
 Calcular1.Enabled:=True;
 ReCargaDatos;
 TabSheet2.Visible:=True;
 PageControl1.ActivePage:=TabSheet2;
 btCreaRed.Enabled:=False;
 MiGrafica;
 Grafica.Graficar(PnGraficar);
end;

procedure TfmMain.btSalirClick(Sender: TObject);
//Cierra la aplicación
var
a:array of array of integer;
begin
 Application.Terminate;
end;

procedure TfmMain.FormCreate(Sender: TObject);
//Crea el formulario, MiCodigos y Grafica
begin
 MiCodigos:=Tcodigos.Create;
 Grafica:=aco.create;
 sgActiviDatos.Visible:=True;
 PageControl1.ActivePage:=TabSheet1;
end;

procedure TfmMain.sgActiviDatosKeyPress(Sender: TObject; var Key: Char);
//Crea una actividad vacía al oprimir <Enter>
begin
 if Key=#13
  then begin
   sgActiviDatos.RowCount:=sgActiviDatos.RowCount+1;
   sgActiviDatos.Col:=1;
   sgActiviDatos.Row:=sgActiviDatos.RowCount-1;
```

```
  sgActiviDatos.Cells[0,sgActiviDatos.RowCount-
1]:=Format('%8d',[sgActiviDatos.Row]);
  btCreaRed.Enabled:=True;
 end;
end;

procedure TfmMain.Antesde1Click(Sender: TObject);
//Calcula la probabilidad de terminar el proyecto antes de su TP
var
 Ok:boolean;
 ik:integer;
 rEstima:TEstima;
 rRC:TRC;
 rParm:TcParm;
begin
 Ok:=InPutQuery('Probabilidad de terminar el proyecto en','Escriba el TP',sTP);
 if Ok
  then begin
   Sigma:=0;
   Mu:=DurPro;
   for ik:=1 to qHol.Items_In do begin {repeat}
    qHol.DeQueue(rRC,SizeOf(rRC));
    qHol.EnQueue(rRC,SizeOf(rRC));{preserva qHol}
    VEstima.Recall(rRC.Index,rEstima);
    Sigma:=Sigma+rEstima.Varianza
   end;{for ik:=1 to qHol.Items_In}
   Sigm:=Sigma;
   Sigma:=Sqrt(Sigma);
   rParm[1]:='1';
   rParm[2]:=FloatToStr((StrToFloat(sTP)-Mu)/Sigma);
   rParm[3]:='1';
   cPert(rParm);
 end;{if Ok then}
end;

procedure TfmMain.Terminarelproyectoenelperiodo1Click(Sender: TObject);
//Calcula la probabilidad de terminar el proyecto en el periodo T1-T2
var
 Ok:boolean;
 ik:integer;
 rEstima:TEstima;
 rRC:TRC;
 rParm:TcParm;
begin
 Ok:=InPutQuery('Probabilidad de terminar el proyecto entre TP1-TP2','Escriba el
TP1',sTP);
 if Ok
```

```
  then begin
   Sigma:=0;
   Mu:=DurPro;
   for ik:=1 to qHol.Items_In do begin {repeat}
    qHol.DeQueue(rRC,SizeOf(rRC));
    qHol.EnQueue(rRC,SizeOf(rRC));{preserva qHol}
    VEstima.Recall(rRC.Index,rEstima);
    Sigma:=Sigma+rEstima.Varianza
   end;{for ik:=1 to qHol.Items_In}
   Sigma:=Sqrt(Sigma);
   rParm[1]:='2';
   rParm[2]:=FloatToStr((StrToFloat(sTP)-Mu)/Sigma);
 end;{if Ok then}
 Ok:=InPutQuery('Probabilidad de terminar el proyecto entre TP1-TP2','Escriba el TP2',sTP);
 if Ok
  then begin
   rParm[3]:=FloatToStr((StrToFloat(sTP)-Mu)/Sigma);
   rParm[4]:='1';
   cPert(rParm);
  end;
end;{Terminarelproyectoenelperiodo1Click}

procedure TfmMain.AlrededordesuTP1Click(Sender: TObject);
//Calcula la probabilidad de terminar el proyecto en su TE±d
var
 Ok:boolean;
 ik:integer;
 rEstima:TEstima;
 rRC:TRC;
 rParm:TcParm;
begin
 Ok:=InPutQuery('Prob. alrededor de su TE','Escriba el ±T',sTP);
 if Ok
  then begin
   Sigma:=0;
   Mu:=DurPro;
   for ik:=1 to qHol.Items_In do begin {repeat}
    qHol.DeQueue(rRC,SizeOf(rRC));
    qHol.EnQueue(rRC,SizeOf(rRC));{preserva qHol}
    VEstima.Recall(rRC.Index,rEstima);
    Sigma:=Sigma+rEstima.Varianza
   end;{for ik:=1 to qHol.Items_In}
   Sigma:=Sqrt(Sigma);
   rParm[1]:='3';
   rParm[2]:=FloatToStr(-(StrToFloat(sTP))/Sigma);
   rParm[3]:=FloatToStr((StrToFloat(sTP))/Sigma);
```

```
   rParm[4]:='2';
   cPert(rParm);
 end;{if Ok then}
end;

procedure TfmMain.Exactamenteen1Click(Sender: TObject);
//Calcula la probabilidad de terminar el proyecto exactamente en T
var
 Ok:boolean;
 ik:integer;
 rEstima:TEstima;
 rRC:TRC;
 rParm:TcParm;
begin
 Ok:=InPutQuery('Prob. término exactamente en T','Escriba el T',sTP);
 if Ok
  then begin
   Sigma:=0;
   Mu:=DurPro;
   for ik:=1 to qHol.Items_In do begin {repeat}
    qHol.DeQueue(rRC,SizeOf(rRC));
    qHol.EnQueue(rRC,SizeOf(rRC));{preserva qHol}
    VEstima.Recall(rRC.Index,rEstima);
    Sigma:=Sigma+rEstima.Varianza
   end;{for ik:=1 to qHol.Items_In}
   Sigma:=Sqrt(Sigma);
   rParm[1]:='4';
   rParm[2]:=FloatToStr(((StrToFloat(sTP))-0.5-Mu)/Sigma);
   rParm[3]:=FloatToStr(((StrToFloat(sTP))+0.5-Mu)/Sigma);
   rParm[4]:='3';
   cPert(rParm);
 end;{if Ok then}
end;

procedure TfmMain.Antesde2Click(Sender: TObject);
//Calcula la probabilidad de no terminar el proyecto antes de su TP
var
 Ok:boolean;
 ik:integer;
 rEstima:TEstima;
 rRC:TRC;
 rParm:TcParm;
begin
 Ok:=InPutQuery('Prob. de no terminar en T','Escriba el T',sTP);
 if Ok
  then begin
   Sigma:=0;
```

```
   Mu:=DurPro;
   for ik:=1 to qHol.Items_In do begin {repeat}
    qHol.DeQueue(rRC,SizeOf(rRC));
    qHol.EnQueue(rRC,SizeOf(rRC));{preserva qHol}
    VEstima.Recall(rRC.Index,rEstima);
    Sigma:=Sigma+rEstima.Varianza
   end;{for ik:=1 to qHol.Items_In}
   Sigma:=Sqrt(Sigma);
   rParm[1]:='1';
   rParm[2]:=FloatToStr(((StrToFloat(sTP))-Mu)/Sigma);
   rParm[3]:='2';
   cPert(rParm);
 end;{if Ok then}
end;

procedure TfmMain.Enelperiodo1Click(Sender: TObject);
//Calcula la probabilidad de no terminar el proyecto dentro de un determinado periodo T1-
T2
var
 Ok:boolean;
 ik:integer;
 rEstima:TEstima;
 rRC:TRC;
 rParm:TcParm;
begin
 Ok:=InPutQuery('Probabilidad de no terminar el proyecto entre TP1-TP2','Escriba el
TP1',sTP);
 if Ok
  then begin
   Sigma:=0;
   Mu:=DurPro;
   for ik:=1 to qHol.Items_In do begin {repeat}
    qHol.DeQueue(rRC,SizeOf(rRC));
    qHol.EnQueue(rRC,SizeOf(rRC));{preserva qHol}
    VEstima.Recall(rRC.Index,rEstima);
    Sigma:=Sigma+rEstima.Varianza
   end;{for ik:=1 to qHol.Items_In}
   Sigma:=Sqrt(Sigma);
   rParm[1]:='2';
   rParm[2]:=FloatToStr((StrToFloat(sTP)-Mu)/Sigma);
 end;{if Ok then}
 Ok:=InPutQuery('Probabilidad de terminar el proyecto entre TP1-TP2','Escriba el
TP2',sTP);
 if Ok
  then begin
   rParm[3]:=FloatToStr((StrToFloat(sTP)-Mu)/Sigma);
   rParm[4]:='4';
```

```
   cPert(rParm);
  end;
end;{Enelperiodo1Click}

procedure TfmMain.Conprobabilidad1Click(Sender: TObject);
//Calcula el tiempo de terminación del proyecto con una probabilidad dada
var
 Ok:boolean;
 ik:integer;
 rEstima:TEstima;
 rRC:TRC;
 rParm:TcParm;
begin
 Ok:=InPutQuery('TP con probabilidad de...','Escriba la probabilidad (%)',sTP);
 if Ok
  then begin
   Sigma:=0;
   Mu:=DurPro;
   for ik:=1 to qHol.Items_In do begin {repeat}
    qHol.DeQueue(rRC,SizeOf(rRC));
    qHol.EnQueue(rRC,SizeOf(rRC));{preserva qHol}
    VEstima.Recall(rRC.Index,rEstima);
    Sigma:=Sigma+rEstima.Varianza
   end;{for ik:=1 to qHol.Items_In}
   Sigma:=Sqrt(Sigma);
   rParm[1]:='5';
   rParm[2]:=FloatToStr((StrToFloat(sTP)/100));
   rParm[3]:=FloatToStr(Mu);
   rParm[4]:=FloatToStr(Sigma);
   rParm[5]:='1';
   cPert(rParm);
 end;{if Ok}
end;

procedure TfmMain.Abrirundocumento1Click(Sender: TObject);
//Abre un archivo pregrabado con una lista de actividades
var
 f:file of TRLin6;
 k,kl:word;
 Lin:TRLin6;
begin
 StatusBar1.SimpleText:='Pulse <Intro> para introducir otra actividad';
 if OpenDialog1.Execute then
  begin
   Resetear;
   AssignFile(f,OpenDialog1.FileName);
   fmMain.Caption:='Pert-'+OpenDialog1.FileName;
```

```
   Reset(f);
   with fmMain.sgActiviDatos do
    begin
     k:=1;
     RowCount:=1;
     while not EOF(f) do
      begin
       Read(f,Lin);
       Lin[0]:=Format('%8d',[StrToInt(Lin[0])]);
       Lin[3]:=Format('%8d',[StrToInt(Lin[3])]);
       Lin[4]:=Format('%8d',[StrToInt(Lin[4])]);
       Lin[5]:=Format('%8d',[StrToInt(Lin[5])]);
       RowCount:=RowCount+1;
       FixedRows:=1;
       for kl:=0 to 6 do
        Rows[k].Strings[kl]:=Lin[kl];
        inc(k);
     end;{while}
    end;{with}
   btCreaRed.Enabled:=True;
   Calcular1.Enabled:=False;
  end;
end;

procedure TfmMain.Guardarundocumento1Click(Sender: TObject);
//Graba en un archivo la lista de actividades actual
var
 f:file of TRLin6;
 k,kl:word;
 Lin:TRLin6;
 S:string;
begin
 if SaveDialog1.Execute then
  begin
   AssignFile(f,SaveDialog1.FileName);
   S:=SaveDialog1.FileName;
   Rewrite(f);
   with fmMain.sgActiviDatos do
    for k:=1 to RowCount-1 do
     begin
      for kl:=0 to 6 do
       Lin[kl]:=Rows[k].Strings[kl];
      Write(f,Lin);
   end;
  end;{if SaveDialog1.Execute then}
end;
```

```
procedure TfmMain.Precedentes1Click(Sender: TObject);
//Marca la opción ver precedentes
begin
 Precedentes1.Checked:=not Precedentes1.Checked;
end;

procedure TfmMain.Ficticias1Click(Sender: TObject);
//Conmuta para hacer visibles o no las actividades ficticias
begin
 Ficticias1.Checked:=not Ficticias1.Checked;
 sgFic.Visible:=Ficticias1.Checked;
end;

{Probabilidad de ocurrencia de un evento dado en un tiempo programado}
procedure TfmMain.probOcurrEvTPClick(Sender: TObject);
//Calcula la probabilidad de que ocurra un determinado evento en un tiempo dado
var
 Ok:boolean;
 nEv,ik:integer;
 iPiv:integer;
 tPxPiv:real;
 rRC:TRC;
 rParm:TcParm;
 rCpm:TRVcpm;
 rEstima:TEstima;
 sEv,sTP:string;
begin {TfmMain.probOcurrEvTPClick}
 if (InPutQuery('Probabilidad de ocurrencia de un evento dado','Escriba el No. del evento',sEv)) then
  if (InPutQuery('Probabilidad de ocurrencia de un evento dado','Escriba el TP',sTP)) then
   begin
    Sigma:=0;
    ik:=0;
    Ok:=False;
    nEv:=StrToInt(sEv);
    repeat {buscar en Vcpm el evento target para determinar la Mu actual}
     inc(ik);
     Vcpm.Recall(ik,rCpm);
     if (rCpm.i=nEv) then
      begin
       Mu:=rCpm.iPx;
       tPxPiv:=rCpm.tPx;{necesario para la búsqueda de la ruta en el siguiente paso}
       Ok:=True;
      end;
    until Ok or not Vcpm.VectorOpStatus;
    repeat {buscar en Vcpm la ruta hasta el evento inicial}
     Ok:=False;
```

```
     ik:=0;
     iPiv:=nEv;
     repeat
      inc(ik);
      Vcpm.Recall(ik,rCpm);
      if (rCpm.j=iPiv)and(rCpm.tPx=tPxPiv) then {encontró el evento en la ruta}
       begin
        VEstima.Recall(rCpm.Index,rEstima);
        Sigma:=Sigma+rEstima.Varianza;
        iPiv:=rCpm.i;{o es rCpm.i ???}
        tPxPiv:=rCpm.iPx;
        ik:=0;
       end;
     until Ok or not Vcpm.VectorOpStatus;
    until iPiv = 1;
    Sigma:=Sqrt(Sigma);
    rParm[1]:='1';
    rParm[2]:=FloatToStr((StrToFloat(sTP)-Mu)/Sigma);
    rParm[3]:='1';
    cPert(rParm);
   end;{if InPutQuery -> sTP then}
end;

procedure TfmMain.FormClose(Sender: TObject; var Action: TCloseAction);
//Cierra el formulario y destruye Grafica y MiCodigos
begin
Grafica.Destroy;
MiCodigos.Destroy;
end;

procedure TfmMain.Vercomplemento1Click(Sender: TObject);
//Conmuta la opción verComplemento
begin
 Opciones1.Checked:=not Opciones1.Checked;
end;

procedure TfmMain.tbtNuevoClick(Sender: TObject);
//Crea una nueva lista de actividades
var
 i,j:integer;
begin
 Caption:='Pert-';
 {StatusBar1.SimpleText:='Pulse <Intro> para introducir otra actividad';}
 with sgActiviDatos do
  begin
   for i:=sgActiviDatos.RowCount-1 downto 1 do
    for j:=0 to 6 do
```

```
    begin
     Rows[i].Strings[j]:='';
    end;
  RowCount:=0;
  RowCount:=sgActiviDatos.RowCount+1;
  FixedCols:=0;
  FixedRows:=1;
  Row:=1;
  Cells[0,1]:=Format('%8d',[1]);
  PageControl1.ActivePage:=TabSheet1;
 end;{with sgActiviDatos do}
end;{TfmMain.ToolButton1Click}

end.
```

```
unit URed;

interface

uses Sysutils,UMain,UDatos;
procedure ArmaRed;
{procedure Pares(Ax:TVActiva);}

implementation


procedure ParCero(var Ax:TActiva);
var
 A1:TVActiva;
begin
 VActiva1.Recall(Ax.Index,A1);
 A1.i:=1;
 VActiva1.Store(A1.Index,A1);
end;{PreCero}

procedure ParUno(var Ax:TActiva);
var
 A1,Ap:TVActiva;
 VPIndex:word;
begin
 VActiva1.Recall(Ax.Index,A1);
 VPIndex:=StrToInt(A1.VP);
 VActiva1.Recall(VPIndex,Ap);
 if(Ap.j=0)
  then begin
   inc(JMax);
   Ap.j:=JMax;
  end;
 A1.i:=Ap.j;
 Ap.Ok:=1; {True;}
 VActiva1.Store(A1.Index,A1);
 VActiva1.Store(Ap.Index,Ap);
end;{PreUno}

procedure ParMas(var Ax:TActiva);
var
 A1,Ap,ATemp,pivAp:TVActiva;
 ColaVP,cVPA,ColaAP,ColaIP,cMi:Queue;
 n,aT,aVP,kAux:word;
 Code:integer;
 fliCero,{indica que aluna actividad del VP tiene i=0 ver líneas 718 y 724}
 flM2,flM3,
```

```
 flP,flDoneCaso2:boolean;
 RFic:TRFic;
 JComun:word;
 S,SVP:string;

procedure IMisma;
begin
 if pivAp.cApar=2
  then begin
   if pivAp.j=0
    then begin
     inc(Jmax);
     pivAp.j:=JMax;
    end;
   if (fliP)
     then begin
      ColaIP.EnQueue(RFic.i,SizeOf(word));
      RFic.i:=pivAp.j;
     end
     else begin
      RFic.i:=pivAp.j;
      fliP:=True;
     end;
  end;
  {else begin}
 kAux:=cVPA.Items_In;
 if kAux>0
  then begin
   repeat
    cVPA.DeQueue(aVP,SizeOf(word));
    VActiva1.Recall(aVP,Ap);
    if (fliP)
     then begin
      ColaIP.EnQueue(RFic.i,SizeOf(word));
      RFic.i:=Ap.j;
     end
     else begin
      RFic.i:=Ap.j;
      fliP:=True;
     end;
    dec(kAux);
   until kAux=0;
  end;
 {end;{else}
 inc(JMax);
 JComun:=JMax;
 if (fliP) then ColaIP.EnQueue(RFic.i,SizeOf(word));
```

```
 repeat
  ColaIP.DeQueue(RFic.i,SizeOf(word));
  RFic.j:=JComun;
  ColaFic.EnQueue(RFic.i,SizeOf(RFic));
 until ColaIP.Is_Empty;
 fliP:=False;
 if pivAp.j=0 then pivAp.j:=JComun;
 pivAp.Ok:=1;
 A1.i:=JComun;
 VActiva1.Store(pivAp.Index,pivAp);
 VActiva1.Store(A1.Index,A1);
 flDoneCaso2:=True;
end;{IMisma}

procedure IDiferente;
begin
if cc32
then begin {presencia de grupos}
    if pivAp.j=0
     then begin
      inc(JMax);
      pivAp.j:=JMax;
     end;
    if (fliP)
     then begin
      ColaIP.EnQueue(RFic.i,SizeOf(word));
      RFic.i:=pivAp.j;
     end
     else begin
      RFic.i:=pivAp.j;
      fliP:=True;
     end;
    pivAp.Ok:=1;
    VActiva1.Store(pivAp.Index,pivAp);
end
else begin {ausencia de grupos}
 Ap:=pivAp;
 if Ap.Ok=0
  then if (Ap.j=0)
   then begin
    inc(JMax);
    Ap.j:=JMax;
    if (fliP)
     then begin
      ColaIP.EnQueue(RFic.i,SizeOf(word));
      RFic.i:=Ap.j;
     end
```

```
     else begin
      RFic.i:=Ap.j;
      fliP:=True;
     end;
    Ap.Ok:=1;
    VActiva1.Store(Ap.Index,Ap)
   end
   else begin
    if (fliP)
     then begin
      ColaIP.EnQueue(RFic.i,SizeOf(word));
      RFic.i:=Ap.j;
     end
     else begin
      RFic.i:=Ap.j;
      {ColaIP.EnQueue(RFic.i,SizeOf(word));{ojo verificar}
      fliP:=True;
     end;
    Ap.Ok:=1;
    VActiva1.Store(Ap.Index,Ap);
   end
  else
   begin  {posiblemente se requiera verificar ficticias pendientes}
    if (fliP)
     then begin
      ColaIP.EnQueue(RFic.i,SizeOf(word));
      RFic.i:=Ap.j;
     end
     else begin
      RFic.i:=Ap.j;
      fliP:=True;
     end;
    Ap.Ok:=1;{con reserva o duda esta línea y la siguiente}
    VActiva1.Store(Ap.Index,Ap)
   end;
   end;{else if cc32}
end;{IDiferente}

procedure Caso2;
var
 fl31,fl32,mismaI,cc3:boolean;
 kAux:integer;
begin
 pivAp:=Ap;
 fl31:=False;
 fl32:=False;
```

```
 for kAux:=1 to cVPA.Items_In do begin {determinar si hay a31 y a32 mediante un carrusel en cVPA}
  cVPA.DeQueue(aVP,SizeOf(aVP));
  VActiva1.Recall(aVP,Ap);
  if Ap.cM3=1 then fl31:=True;{levanta fl31 si hubiera a31}
  if Ap.cM3=2 then fl32:=True;{levanta fl32 si hubiera a32}
  cVPA.EnQueue(aVP,SizeOf(aVP));
 end;{for kAux:=1 to cVPA.Items_In}
 if (pivAp.cM3=0)or((pivAp.cM3=2)and not(fl31 and fl32))
 then begin
  mismaI:=False;
  cc3:=False;
  {pivAp:=Ap;}
  kAux:=cVPA.Items_In;
  if kAux>0
   then begin
    repeat
     cVPA.DeQueue(aVP,SizeOf(word));
     cVPA.EnQueue(aVP,SizeOf(word));
     VActiva1.Recall(aVP,Ap);
     if (Ap.cApar=3){or(pivAp.cApar=3)}
      then begin
       cc3:=True;
       cc32:=Ap.cM3=2; {revisar que al final quede el valor correcto para cc32}
      end;
     if pivAp.i=Ap.i then mismaI:=True;
     dec(kAux);
    until kAux=0;
   end;
  if not(cc3)and(mismaI)
   then IMisma
   else IDiferente;
 end;{(pivAp.cM3=0)or((pivAp.cM3=2)and(fl31 or fl32))}
end;{Caso2}

procedure Caso3;
var
 m,k,JTemp,MaxJ:word;
begin
 if (Ap.j=0)
  then begin
   if (Ap.cApar=3) then inc(JMax);
   Ap.j:=JMax;
   VActiva1.Store(Ap.Index,Ap);
  end;
 MaxJ:=Ap.j;
 k:=ColaVP.Items_In;
```

```
 if (k<1)
  then begin
   Ap.Ok:=1;
   A1.i:=Ap.j;
   VActiva1.Store(Ap.Index,Ap);
   VActiva1.Store(A1.Index,A1);
  end
  else for m:=1 to k do
   begin
    ColaVP.DeQueue(aT,SizeOf(word));
    VActiva1.Recall(aT,ATemp);
    if (MaxJ<Atemp.j)
     then begin
      MaxJ:=ATemp.j;
      JTemp:=Ap.j;
      Ap.j:=ATemp.j;
      ATemp.j:=JTemp;
      VActiva1.Store(Ap.Index,Ap);
      VActiva1.Store(Atemp.Index,Atemp);
     end;
    if (ATemp.cApar=3)
     then begin
      ATemp.j:=JMax;
      ATemp.Ok:=1;
      VActiva1.Store(Atemp.Index,Atemp);
     end
     else ColaVP.EnQueue(aVP,SizeOf(word));
    Ap.Ok:=1;
    A1.i:=Ap.j;
    VActiva1.Store(Ap.Index,Ap);
    VActiva1.Store(A1.Index,A1);
   end;
 if fliP
  then begin
   RFic.j:=Ap.j;
   fliP:=False;
   ColaIP.EnQueue(RFic.i,SizeOf(word));
   repeat
    ColaIP.DeQueue(RFic.i,SizeOf(word));
    ColaFic.EnQueue(RFic,SizeOf(RFic));
   until ColaIP.Is_Empty;
  end;
  JMax:=MaxJ;
end;{Caso3}

procedure jPendiente;
begin
```

```
 if (Ap.j=0)
  then begin
   inc(JMax);
   Ap.j:=JMax;
  end;
 if fliP
  then begin
   RFic.j:=Ap.j;
   fliP:=False;
   ColaIP.EnQueue(RFic.i,SizeOf(word));
   repeat
    ColaIP.DeQueue(RFic.i,SizeOf(word));
    ColaFic.EnQueue(RFic,SizeOf(RFic));
   until ColaIP.Is_Empty;
  end;
 Ap.Ok:=1;
 VActiva1.Store(Ap.Index,Ap);
 A1.i:=Ap.j;
 VActiva1.Store(A1.Index,A1);
end;{jPendiente}

procedure M1;
begin
 repeat
  ColaVP.DeQueue(aVP,SizeOf(word));
  VActiva1.Recall(aVP,Ap);
  case Ap.cApar of
   1,2:Caso2;
     3:Caso3;
  end;{case}
 until ColaVP.Is_Empty;
end;{M1}

procedure SolPivote(Ap:TActiva);
begin

end;{SolPivote}

procedure SolMi(cMi:Queue;Ax:TActiva);
var
 aPri,aSeg:TActiva;
 {A1,}APrimera,ASegunda:TVActiva;
 MaxJ:word;
begin
 cMi.Dequeue(aPri,SizeOf(aPri));
 VActiva1.Recall(aPri.Index,APrimera);
 if APrimera.j=0
```

```
  then begin
  inc(JMax);
  APrimera.j:=JMax;
  end;
 while not(cMi.Is_Empty) do
  begin
   cMi.DeQueue(aSeg,SizeOf(aSeg));
   VActiva1.Recall(aSeg.Index,ASegunda);
   if ASegunda.j=0
    then begin
     inc(JMax);
     ASegunda.j:=JMax;
    end;
   if APrimera.j<ASegunda.j
    then begin
     RFic.i:=APrimera.j;
     MaxJ:=ASegunda.j;
     APrimera.Ok:=1;
     VActiva1.Store(APrimera.Index,APrimera);
     APrimera:=ASegunda;
    end
    else begin
     RFic.i:=ASegunda.j;
     MaxJ:=APrimera.j;
     ASegunda.Ok:=1;
     VActiva1.Store(ASegunda.Index,ASegunda);
    end;
   ColaIP.EnQueue(RFic.i,SizeOf(word));
  end;{while}
  RFic.j:=MaxJ;
  APrimera.Ok:=1;
  VActiva1.Store(APrimera.Index,APrimera);
 repeat
  ColaIP.DeQueue(RFic.i,SizeOf(word));
  ColaFic.EnQueue(RFic,SizeOf(RFic));
 until ColaIP.Is_Empty;
end;{SolMi}

procedure M2;
var
 k,{ki,}kJ:word;
 pivA:TActiva;
begin
 k:=ColaVP.Items_In;
 cMi.Init(SizeOf(TActiva),k);
 cVPA.Clear;
 cVPA:=ColaVP;
```

```
 while not(cVPA.Is_Empty) do
  begin
   cVPA.DeQueue(pivA,SizeOf(pivA));
   VActiva1.Recall(pivA.Index,pivAp);
   k:=cVPA.Items_In;
   repeat
    cVPA.DeQueue(Ap.Index,SizeOf(Ap.Index)); {revisar porque Ap?}
    VActiva1.Recall(Ap.Index,A1);{revisar porque Ap con A1?}
    if pivAp.i<>A1.i
     then cVPA.EnQueue(Ap,SizeOf(Ap))
     else cMi.EnQueue(Ap.Index,SizeOf(Ap.Index));
    dec(k);
   until (k=0);
   if(cMi.Is_Empty)
    then begin
     cVPA.EnQueue(pivA,SizeOf(pivA))
     {SolPivote(Ax);}
    end
    else begin
     cMi.EnQueue(pivA.Index,SizeOf(pivA.Index));
     SolMi(cMi,Ax);
    end;
  end;{while}
 k:=cVPA.Items_In;
 repeat  {revisar si repeat debe ser while porque k puede ser cero, caso test06}
  cVPA.DeQueue(Ap,SizeOf(Ap));
  kJ:=Ap.j;
  cVPA.Enqueue(Ap,SizeOf(Ap));{Error se encola Ap cuando debe ser Ap.Index-!!revisar
estado de cVPA}
 until ((k=0) or (kJ>0));
 while not(cVPA.Is_Empty) do
  begin
   cVPA.DeQueue(Ap,SizeOf(Ap));
   Ap.j:=kJ;
   Ap.Ok:=1;
   VActiva1.Store(Ap.Index,Ap);
  end;{while}
 VActiva1.Recall(Ax.Index,A1);
 A1.i:=JMax;
 VActiva1.Store(A1.Index,A1);
end;{M2}

procedure M3S;
var
 jComun:word;
begin
 jComun:=0;
```

```
if pivAp.j<>0 then jComun:=pivAp.j;
kAux:=ColaIp.Items_In;
if kAux=0 then kAux:=1;{esto evita que el siguiente ciclo se repita 65554 veces innecesariamente}
if jComun=0 then
 repeat
  ColaAP.DeQueue(aVP,SizeOf(aVP));
  VActiva1.Recall(aVP,Ap);
  if (Ap.j<>0)and(Ap.cApar<>2) then jComun:=Ap.j;
  ColaAP.EnQueue(aVP,SizeOf(aVP));
  dec(kAux);
 until(jComun<>0)or(kAux=0);

if jComun=0 {calcula la j de la pivAp}
 then begin
  inc(jMax);
  jComun:=jMax;
 end;
pivAp.j:=jComun;
pivAp.Ok:=1;
VActiva1.Store(pivAp.Index,pivAp);

if flM2
 then begin
  inc(JMax);
  JComun:=JMax;
  if (fliP) then ColaIP.EnQueue(RFic.i,SizeOf(word));
  RFic.i:=pivAp.j;
  ColaIP.EnQueue(RFic.i,SizeOf(word));
  repeat
   ColaIP.DeQueue(RFic.i,SizeOf(word));
   RFic.j:=JComun;
   ColaFic.EnQueue(RFic,SizeOf(RFic));
  until ColaIP.Is_Empty;
  fliP:=False;
  Ap.j:=JComun;
  Ap.Ok:=1;
  A1.i:=JComun;
  VActiva1.Store(Ap.Index,Ap);
  VActiva1.Store(A1.Index,A1);
 end
 else begin
while not ColaAP.Is_Empty do {pone la misma j a las A31 con i diferente}
 begin
  ColaAP.DeQueue(aVP,SizeOf(aVP));
  VActiva1.Recall(aVP,Ap);
  if Ap.j=0 then Ap.j:=jComun;
```

```
   Ap.Ok:=1;
   VActiva1.Store(Ap.Index,Ap);
  end;{while not ColaAP.Is_Empty}
 if fliP
  then begin
   RFic.j:=JComun;
   fliP:=False;
   ColaIP.EnQueue(RFic.i,SizeOf(word));
   repeat
    ColaIP.DeQueue(RFic.i,SizeOf(word));
    ColaFic.EnQueue(RFic,SizeOf(RFic));
   until ColaIP.Is_Empty;
  end;
 A1.i:=jComun;
 VActiva1.Store(A1.Index,A1);
 end;

end;{M3S}

function gpoVP(gS,VP:ShortString):boolean;
var
 ColaGpos,ColaVP:Queue;
 k,nk:integer;
 gElem,elemVP:TStr4;
 rGpos:TRGpos;
 Ok:boolean;
begin
 ColaGpos.Init(SizeOf(TRGpos),100);
 ColaVP.Init(SizeOf(TStr4),100);
 Ok:=True; {utilizada para la verificación final si el grupo está en VP}
 gElem:='';
 for nk:=1 to Length(gS) do {separa los elementos del grupo}
  if gS[nk]<>','
   then gElem:=gElem+gS[nk]
   else begin
    rGpos.Cad:=gElem;
    rGpos.Si:=False;
    ColaGpos.EnQueue(rGpos,SizeOf(rGpos));
    gElem:='';
   end;
 elemVP:='';
 for nk:=1 to Length(VP) do {separa los elementos del VP}
  if VP[nk]<>','
   then elemVP:=elemVP+VP[nk]
   else begin
    ColaVP.EnQueue(elemVP,SizeOf(TStr4));
    elemVP:='';
```

```
  end;
 for nk:=1 to ColaGpos.Size do {carrusel del grupo para verificar si está en VP}
  begin
   ColaGpos.DeQueue(rGpos,SizeOf(rGpos));
   for k:=1 to ColaVP.Size do {carrusel de VP}
    begin
     ColaVP.DeQueue(elemVP,SizeOf(elemVP));
     ColaVP.EnQueue(elemVP,SizeOf(elemVP));
     if rGpos.Cad=elemVP then rGpos.Si:=True;
    end;
   ColaGpos.EnQueue(rGpos,SizeOf(rGpos));
  end;
 for nk:=1 to ColaGpos.Size do {verificación final}
  begin
   ColaGpos.DeQueue(rGpos,SizeOf(rGpos));
   Ok:=Ok and rGpos.Si;
  end;
 gpoVP:=Ok;
end;{gpoVP}

procedure M3R;
var
 JComun:word;
 kG,{nG,}gk,kAux:integer;
 fl31,flGpo{,flOk}:boolean;
 gS,gSn:ShortString;
 {nk:word;}
begin
 JGpo.Init(Gpo.VSize);
 JComun:=0;
 kG:=0;
 gk:=0;
 for kAux:=1 to JGpo.MaxVectorSize do {inicializa el arreglo las J's de los grupos en cero}
  JGpo.Store(kAux,0);
 for kAux:=1 to cVPA.Items_In do begin {determinar si hay a31 mediante un carrusel en cVPA}
  cVPA.DeQueue(aVP,SizeOf(aVP));
  VActiva1.Recall(aVP,Ap);
  if Ap.cM3=1 then fl31:=True; {levanta fl31 si hubiera a31}
  cVPA.EnQueue(aVP,SizeOf(aVP));
 end;{for kAux:=1 to cVPA.Items_In}

 repeat {comprueba que un Gpo esté contenido en A1.VP}
  inc(gk);
  Gpo.Recall(gk,gS);
  flGpo:=gpoVP(gS,A1.VP);{se levanta la bandera si el Gpo está contenido}
  {flGpo:=pos(gS,A1.VP)<>0;}
```

```
if flGpo {el grupo si está contenido}
 then begin
  inc(kG);
  gSn:='';
  for kAux:=1 to Length(gS) do {calcula la j del Gpo}
   if gS[kAux]<>','
    then gSn:=gSn+gs[kAux]
    else begin
     VActiva1.Recall(StrToInt(gSn),Ap);
     if (Ap.j>0)and(Ap.cApar=3)
      then JGpo.Store(gk,Ap.j);
     gSn:='';
    end; {else}
   VActiva1.Recall(StrToInt(gSn),Ap);
   if Ap.j>0 then JGpo.Store(gk,Ap.j);
   JGpo.Recall(gk,JComun);
   if JComun=0
    then begin
     inc(JMax);
     JGpo.Store(gk,JMax);
    end;
  gSn:='';
  for kAux:=1 to Length(gS) do {procesa las A32 del Gpo}
   if gS[kAux]<>','
    then gSn:=gSn+gs[kAux]
    else begin
     VActiva1.Recall(StrToInt(gSn),Ap);
     if Ap.j=0
      then JGpo.Recall(gk,Ap.j);
     Ap.Ok:=1;
     VActiva1.Store(Ap.Index,Ap);
     gSn:='';
  end;{else + for kAux:=1 to Length(gS)}
  VActiva1.Recall(StrToInt(gSn),Ap);
  JGpo.Recall(gk,Ap.j);
  Ap.Ok:=1;
  VActiva1.Store(Ap.Index,Ap);
  if fliP
   then begin
    ColaIP.EnQueue(RFic.i,SizeOf(word));
    JGpo.Recall(gk,RFic.j);
    repeat
     ColaIP.DeQueue(RFic.i,SizeOf(word));
     ColaFic.EnQueue(RFic,SizeOf(RFic));
    until ColaIP.Is_Empty;
    fliP:=False;
   end {then}
```

```
     else begin
      JGpo.Recall(gk,RFic.i);
     end;{else}
   end;{if flGpo (kG>0)}
  {flGpo:=False;}
 until (gk=Gpo.VSize);

 if fl31 then {si hubiera A31, genera la ficticia por c/gpo}
  begin
   for gk:=1 to JGpo.VSize do begin
    JGpo.Recall(gk,JComun);
    if JComun>0 then begin
     if fliP then ColaIP.EnQueue(RFic.i,SizeOf(word));
     rFic.i:=JComun;
     fliP:=True;
    end;{if JComun>0}
   end;{for gk:=1 to JGpo.VSize}
   JComun:=0;
   for kAux:=1 to cVPA.Items_In do begin {calcula la JComun para las 31 en carrusel}
    cVPA.DeQueue(aVP,SizeOf(aVP));
    VActiva1.Recall(aVP,Ap);
    if (Ap.j>0)and(Ap.cM3=1) then JComun:=Ap.j;
    cVPA.EnQueue(aVP,SizeOf(aVP));
   end;{for kAux:=1 to cVPA.Items_In}
   if JComun=0 then begin
    inc(JMax);
    JComun:=JMax;
   end;
   for kAux:=1 to cVPA.Items_In do begin {procesa las A31 utilizando la JComun en
carrusel}
    cVPA.DeQueue(aVP,SizeOf(aVP));
    VActiva1.Recall(aVP,Ap);
    if Ap.cM3=1 then
     begin
      Ap.j:=JComun;
      Ap.Ok:=1;
      VActiva1.Store(Ap.Index,Ap);
     end;
    cVPA.EnQueue(aVP,SizeOf(aVP));
   end;{for kAux:=1 to cVPA.Items_In}
   if fliP
    then ColaIP.EnQueue(RFic.i,SizeOf(word));{encola la última ficticia pendiente}
   repeat {procesa todas las ficticias pendientes}
    ColaIP.DeQueue(RFic.i,SizeOf(word));
    RFic.j:=JComun;
    ColaFic.EnQueue(RFic,SizeOf(RFic));
   until ColaIP.Is_Empty;
```

```
   fliP:=False;{registra que ya no hay ficticias pendientes}
   A1.i:=JMax;
   VActiva1.Store(A1.Index,A1);
  end;{if fl31}

 if (not fl31)and(kG=1) then {procesa A1.i}
  begin
   for gk:=1 to JGpo.VSize do begin
    JGpo.Recall(gk,JComun);
    if JComun>0 then begin
     A1.i:=JComun;
     VActiva1.Store(A1.Index,A1);
    end;{if JComun>0}
   end;{for gk:=1 to JGpo.VSize}
  end;{if (not fl31)and(flGpo)and(kG=1)}

  if (not fl31)and(kG>1) then {procesa las ficticias por haber mas de un gpo en VP}
   begin
    for gk:=1 to JGpo.VSize do begin
     JGpo.Recall(gk,JComun);
     if JComun>0 then begin
      if fliP then ColaIP.EnQueue(RFic.i,SizeOf(word));
      rFic.i:=JComun;
      fliP:=True;
     end;{if JComun>0}
    end;{for gk:=1 to JGpo.VSize}
    inc(JMax);
    JComun:=JMax;
    if fliP then ColaIP.EnQueue(RFic.i,SizeOf(word));{encola la última ficticia pendiente}
    repeat {procesa todas las ficticias pendientes}
     ColaIP.DeQueue(RFic.i,SizeOf(word));
     RFic.j:=JComun;
     ColaFic.EnQueue(RFic,SizeOf(RFic));
    until ColaIP.Is_Empty;
    fliP:=False;{registra que ya no hay ficticias pendientes}
    A1.i:=JComun;
    VActiva1.Store(A1.Index,A1);
   end;{if (not fl31)and(flGpo)and(kG>1)}
 JGpo.Done;
end;{M3R}

procedure M3;{GrupoMj;}
var
 flM3r:boolean;
begin
 if (Ap.Ok=0)or(Ap.cM3=2)
  then begin
```

```
   flM3r:=False;
   cVPA:=ColaVP;
   kAux:=cVPA.Items_In;
   while {not(flM3r)and}(kAux>0) do
    begin
     cVPA.DeQueue(aVp,SizeOf(aVP));
     VActiva1.Recall(aVp,Ap);
     flM3r:=Ap.cM3=2;
     cVPA.EnQueue(Ap.Index,SizeOf(Ap.Index));
     dec(kAux);
    end;{while}
   if flM3r
    then M3R
    else M3S;
  end{if Ap.Ok=0 then}
  else if A1.i=0
   then begin
    A1.i:=JMax;
    VActiva1.Store(A1.Index,A1);
    if (fliP)
     then ColaIP.EnQueue(RFic.i,SizeOf(word))
     else begin
      if RFic.i=62852 then RFic.i:=JMax-1;
      RFic.j:=JMax;
      ColaFic.EnQueue(RFic,SizeOf(RFic));
      fliP:=False;{verificar que funcione para todos los casos, se puso por A1=I en M3-01}
     end;
   end;
end;{M3}

procedure M1oM2;
begin
 if (Ap.Ok=1)and(A1.i=0)
  then if fliP
   then begin
    inc(JMax);
    JComun:=JMax;
    ColaIP.EnQueue(RFic.i,SizeOf(word));
    repeat {procesa todas las ficticias pendientes}
     ColaIP.DeQueue(RFic.i,SizeOf(word));
     RFic.j:=JComun;
     ColaFic.EnQueue(RFic,SizeOf(RFic));
    until ColaIP.Is_Empty;
    fliP:=False;
    A1.i:=JComun;
    VActiva1.Store(A1.Index,A1);
   end
```

```
   else begin
    A1.i:=Ap.j;
    VActiva1.Store(A1.Index,A1);
   end;
 if Ap.Ok=0
  then if flM2
   then M2
   else M1;
end;{M1oM2}

{procedure iFaltante;
var
 tempColaVP,
 tempcVPA,
 tempColaAP,
 tempColaIP:Queue;
 tempA1,
 tempAp,
 temppivAp:TVActiva;
 tempaVP,
 tempkAux:word;
 tempflM2,
 tempflM3,
 tempflP,
 tempflDoneCaso2:boolean;
 tempRFic:TRFic;
 tempJComun:word;
 i:integer;
 j:word;
begin
 tempColaVP.Init(SizeOf(word),ColaVP.Size);
 tempcVPA.Init(SizeOf(word),cVPA.Size);
 tempColaAP.Init(SizeOf(word),ColaAP.Size);
 tempColaIP.Init(SizeOf(word),ColaIP.Size);
 for i:=1 to ColaVP.Size do begin
  ColaVP.DeQueue(j,SizeOf(j));
  tempColaVP.EnQueue(j,SizeOf(j));
 end;
 for i:=1 to ColaVP.Size do begin
  cVPA.DeQueue(j,SizeOf(j));
  tempcVPA.EnQueue(j,SizeOf(j));
 end;
 for i:=1 to ColaVP.Size do begin
  ColaAP.DeQueue(j,SizeOf(j));
  tempColaAP.EnQueue(j,SizeOf(j));
 end;
 for i:=1 to ColaIP.Size do begin
```

```
  ColaIP.DeQueue(j,SizeOf(j));
  tempColaIP.EnQueue(j,SizeOf(j));
 end;
 ColaVP.Clear;
 cVPA.Clear;
 ColaAP.Clear;
 ColaIP.Clear;
 tempkAux:=kAux;
 tempflM2:=flM2;
 tempflM3:=flM3;
 tempflP:=flP;
 tempflDoneCaso2:=flDoneCaso2;
 tempRFic:=RFic;
 tempJComun:=JComun;
 if pivAp.i=0
  then begin
   tempcVPA.EnQueue(Ap.Index,SizeOf(Ap.Index));
   Pares(pivAp);
   VActiva1.Recall(pivAp.Index,pivAp);
  end
  else begin
  end;
end;{iFaltante}

begin {ParMas}
 flDoneCaso2:=False;{utilizada en caso2 con IMisma}
 VActiva1.Recall(Ax.Index,A1);
 ColaVP.Init(SizeOf(word),A1.LVP);
 cVPA.Init(SizeOf(word),A1.LVP);
 ColaIP.Init(SizeOf(word),1000);
 ColaAP.Init(SizeOf(word),1000);
 flP:=False;
 flM2:=False;
 flM3:=False;
 fliCero:=False;
 SVP:=A1.VP;
 for n:=1 to A1.LVP-1 do
  begin
   S:=Copy(SVP,1,Pos(',',SVP)-1);
   Val(S,aVP,Code);
   ColaVP.EnQueue(aVP,SizeOf(word));
   cVPA.EnQueue(aVP,SizeOf(word));
   Delete(SVP,1,Length(S)+1);
  end;
 for n:=1 to cVPA.Items_In do
  begin
   cVPA.DeQueue(aVP,SizeOf(aVP));
```

```
    VActiva1.Recall(aVP,Ap);
    flM3:=Ap.cApar=3;
    cVPA.EnQueue(aVP,SizeOf(aVP));
   end;
  Val(SVP,aVP,Code);
  ColaVP.EnQueue(aVP,SizeOf(word));
  cVPA.EnQueue(aVP,SizeOf(word));
  while not(cVPA.Is_Empty) do
   begin
    cVPA.DeQueue(aVP,SizeOf(word));
    VActiva1.Recall(aVP,Ap);
    if (Ap.cApar=1) or (Ap.cApar=2)
     then Caso2
     else {begin}
      if cVPA.Is_Empty
       then jPendiente
       else begin
        fliCero:=Ap.i=0;{para ejecutar IFaltante}
        pivAp:=Ap; {revisar si es la última de cVPA}
        kAux:=cVPA.Items_In;
        repeat
         cVPA.DeQueue(aVP,SizeOf(word));
         VActiva1.Recall(aVP,Ap);
         fliCero:=Ap.i=0;{para ejecutar IFaltante}
         flM2:=pivAp.i=Ap.i;
         if not flM2 then
          begin
           if fliCero then flP:=True; {iFaltante;}
           ColaAP.EnQueue(aVP,SizeOf(word));
           VActiva1.Recall(aVP,Ap);
           VActiva1.Store(Ap.Index,Ap);
          end;
         dec(kAux);
        until(flM2 or (kAux=0));
      end;{if cVPA.Is_Empty}
   end;{while}
   {nuevo código}
   if not flDoneCaso2 then
   if (flP or flM3)
    then M3 {GrupoMj}
    else M1oM2;
end;{ParMas}

procedure Pares(Ax:TActiva);
var
 A1:TVActiva;
begin
```

```
 VActiva1.Recall(Ax.Index,A1);
 if A1.Ok=0
  then if ((A1.cApar=0) and (A1.i>0))
   then begin
      if not(flJMax)
       then begin
        inc(JMax);
        flJMax:=True;
       end;
      VActiva1.Recall(Ax.Index,A1);
      A1.j:=JMax;
      A1.Ok:=1;
      VActiva1.Store(A1.Index,A1);
     end
   else
   case A1.LVP of
    0:ParCero(Ax);
    1:ParUno(Ax);
    else ParMas(Ax);
   end;{case}
  {end;}
end;{Pares}

procedure ArmaRed;
var
 Ax:TActiva;
 A1,Ap:TVActiva;
begin
 fliP:=False;
 flJMax:=False;
 JMax:=1;
 ColaFic.Init(SizeOf(TRFic),1000);
 if not(Cola1.Is_Empty)
  then begin
   Cola1.DeQueue(Ax,SizeOf(Ax));
   VActiva1.Recall(Ax.Index,A1);
   repeat
    if (A1.cR>0){and(A1.i>1){and(A1.cApar>0)}
     then begin
      VActiva1.Recall(A1.cR,Ap);
      A1.i:=Ap.i;
      VActiva1.Store(A1.Index,A1);
      if (Cola1.Items_In<nT)
      then begin
      if not(flJMax)
       then begin
        inc(JMax);
```

```
        flJMax:=True;
       end;
      VActiva1.Recall(Ax.Index,A1);
      A1.j:=JMax;
      A1.Ok:=1;
      VActiva1.Store(A1.Index,A1);
     end;
     end
     else Pares(Ax);
    VActiva1.Recall(Ax.Index,A1);
    if A1.Ok=0
     then Cola1.EnQueue(Ax,SizeOf(Ax));
    Cola1.DeQueue(Ax,SizeOf(Ax));
    VActiva1.Recall(Ax.Index,A1);
   until Cola1.Is_Empty;
   Pares(Ax);
  end;
end;{ArmaRed}
end.
```

```
unit MiUnit ;

interface
uses
 Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
 Grids, StdCtrls, Udatos;

Type

TCodigos=class
 ElemVp : array of array of integer; //CONTIENE A VP EN ENTEROS
 CodApa : array of array of integer;
 GruposA : array of array of integer;
 MiDondeGrupos:array of array of integer;  //DICE DONDE ESTAN LOS GRUPOS
 CR1 : array of integer;   //CODIGO DE REPETICION
 procedure VerifComas;//ME ARREGLA A VP Y ME DICE SI ESTA MAL
 function GuardarDatos:integer;     //VERIFICA QUE LOS ELEMENTOS NO
SOBREPASES LOS LIMITES Y LOS GUARDA //REGRESA 0 SI ESTA BIEN 1 SI
ESTA MAL
 procedure CodAparicion;        //GENERA LA TABLA Y DICE SI VIENE SOLA O
ACOMPAÑADA
 procedure Cm32;
 procedure CrearMiDondeEstan;
 procedure GuardaCM32;
 procedure Ordenar;
 procedure MiOrden;
 procedure OrdenVP;
 procedure CR;
 Function a(X:integer):Integer;
 procedure borrar;
end;

var
MiCodigos:TCodigos;
{****************************************************************}
implementation

procedure TCodigos.CR;
var
 c,I,J:integer;
 eso:TVactiva;
 K, L :integer;
 cont:integer;
 StrGrupo:string[30];
begin
 SetLength(Cr1,VActiva1.MaxVectorSize+1);
 for I:=0 to VActiva1.MaxVectorSize-1 do
```

```
 Cr1[I]:=0;
cont:=0;
for I:=1 to High(MiCodigos.ElemVp)-1 do
 if Cr1[I]=0
 then begin
  for J:=I+1 to High(MiCodigos.ElemVp) do
   begin
    if Cr1[J]=0 then
     if High(MiCodigos.ElemVp[I])=High(MiCodigos.ElemVp[J])
      then begin
       for K:=1 to High(MiCodigos.ElemVp[I]) do
        for L:=1 to High(MiCodigos.ElemVp[I]) do
         if MiCodigos.ElemVp[I,K]=MiCodigos.ElemVp[J,L] then cont:=cont+1;
       c:=High(MiCodigos.ElemVp[I]) ;
       c:=High(MiCodigos.ElemVp[I]) ;
       if cont=High(MiCodigos.ElemVp[I]) then Micodigos.CR1[J] := I;
      end;{if High(MiCodigos.ElemVp[I])=High(MiCodigos.ElemVp[J]) then...}
      cont:=0;
   end;{for J:=I+1 to High(MiCodigos.ElemVp)}
 end;{for I:=1 to High(MiCodigos.ElemVp)-1}

for I:=1 to High(MiCodigos.ElemVp)do
 begin
  Vactiva1.Recall (I, eso);
  eso.cr:= CR1[I];
  Vactiva1.Store (I, eso);
 end;{for I:=1 to High(MiCodigos.ElemVp)}


 gpo.MaxVectorSize :=High(GruposA)+1;

 for I:=0 to High(GruposA)do
  begin
   strgrupo:='';
   for J:=0 to High(GruposA[I])do
    begin
     if j< High(GruposA[I])
      then strgrupo:=strgrupo + IntToStr(GruposA[I,J]) + ','
      else strgrupo:= strgrupo + IntToStr (GruposA[I,J]);
    end;{for J:=0 to High(GruposA[I])}
    Gpo.Store(I+1, strgrupo);
  end;{for I:=0 to High(GruposA)}


end;{TCodigos.CR}
```

```
{**************************************************************}
{**************************************************************}

procedure TCodigos.OrdenVP;
var
 I,j,k,l:integer;   //AQUI YA TRAIGO EN MI VP
 A,B,C:integer;
 C320 : array of integer;
 Auxiliar : array of array of integer;
 eso, eso1, esoAux:TVActiva;
 VActiva2:AVector;
begin
 SetLength(Auxiliar,High(ElemVp)+1);
 SetLength(C320,High(ElemVp)+1);

 for I:=0 to High(ElemVp) do C320[I]:=1;
 for I:=0 to High(ElemVp) do //AQUI ALMACENO EN AUXILIAR A TODOS LOS DE
CAPAR=1
  begin                         //DE TODOS LOS VP
   SetLength(Auxiliar[I],High(ElemVp[I])+1);
   A:=0;
   for J:=0 to High(ElemVp[I]) do
    begin
     Vactiva1.Recall(ElemVp[I,J],eso);
     if eso.cApar=1
      then begin
       Auxiliar[I,c320[I]]:=ElemVp[I,J];
       c320[I]:=c320[I]+1;
      end;{if eso.cApar=1 then}
    end;{for J:=0 to High(ElemVp[I])}
  end;{for I:=0 to High(ElemVp)}

 for I:=0 to High(ElemVp) do  //AQUI ALMACENO EN AUXILIAR A TODOS LOS DE
CAPAR=2
  begin                         //Y CM3=0
   for J:=0 to High(ElemVp[I]) do
    begin
     Vactiva1.Recall(ElemVp[I,J],eso);
     if (eso.cApar=2) and (eso.cM3=0)
      then begin
       Auxiliar[I,c320[I]]:=ElemVp[I,J];
       c320[I]:=c320[I]+1;
      end;{if (eso.cApar=2) and (eso.cM3=0) then}
    end;{for J:=0 to High(ElemVp[I])}
  end;{for I:=0 to High(ElemVp)}
```

```
 for I:=0 to High(ElemVp) do        //AQUI ALMACENO EN AUXILIAR A TODOS LOS
DE CAPAR=2
  begin                        //Y CM3=2
   for J:=0 to High(ElemVp[I]) do
    begin
     Vactiva1.Recall(ElemVp[I,J],eso);
     if (eso.cApar=2) and (eso.cM3=2)
      then begin
       Auxiliar[I,c320[I]]:=ElemVp[I,J];
       c320[I]:=c320[I]+1;
      end;{if (eso.cApar=2) and (eso.cM3=2) then}
    end;{for J:=0 to High(ElemVp[I])}
 end;{I:=0 to High(ElemVp)}

 for I:=0 to High(ElemVp) do        //AQUI ALMACENO EN AUXILIAR A TODOS LOS
DE CAPAR=3
  begin                        //Y CM3=1
   for J:=0 to High(ElemVp[I]) do
    begin
     Vactiva1.Recall(ElemVp[I,J],eso);
     if (eso.cApar=3) and (eso.cM3=1)
      then begin
       Auxiliar[I,c320[I]]:=ElemVp[I,J];
       c320[I]:=c320[I]+1;
      end;{if (eso.cApar=3) and (eso.cM3=1) then}
    end;{for J:=0 to High(ElemVp[I])}
  end;{for I:=0 to High(ElemVp)}

//AQUI SE PONEN LOS GRUPOS, O SEA EL CODIGO 32
 for I:=1 to high(ElemVp) do //con este paso entre los elementos del vp
  begin
   if High(MiDondeGrupos[I])>=0  //AQUI VEO SI HAY GRUPOS
    then begin
     B:=1;
     for A:=1 to High(Auxiliar[I]) do
      if Auxiliar[I,A]<>0 then inc(B);
        //esto lo analizas           //j
     for J:=0 to High(MiDondeGrupos[I]) do //ES LA QUE ME DICE QUE Y CUANTOS
GRUPOS TIENE
      begin
       for K:=0 to High ( GruposA[MiDondeGrupos[I,J]-1]) do //PARA PASAR EN LOS
ELEMENTOS DEL GRUPO
        begin
         for L:=1 to High(Auxiliar[I]) do  //PARA PASAR ENTRE LOS ELEMENTOS YA
GUARDADOS. Si no he guardado el elemento //ESTA PARTE ESTA MAL
           if Auxiliar[I,L] = GruposA[MiDondeGrupos[I,J]-1,k] then C:= 1; //ya esta en el vp
         if C=0 //SI NO ESTA EN EL VP LO PONE
```

```
          then begin
           Auxiliar[I,B]:= GruposA[MiDondeGrupos[I,J]-1,K];
           inc(B);
          end;{if C=0 then}
         C:=0;
        end;{for K:=0 to High ( GruposA[MiDondeGrupos[I,J]-1])}
      end;{for J:=0 to High(MiDondeGrupos[I])}
    end;{if High(MiDondeGrupos[I])>=0 then}
  end;{for I:=1 to high(ElemVp)}

//YA NO LO TENGO QUE PONER    O MAS BIEN SI PARA GUARDAR LA NUEVA
CADENA
 for I:=1 to high(Micodigos.ElemVp) do //PARA MODIFICAR A LOS VP
  begin
   VActiva1.Recall(I,Eso);
   Eso.VP:='';
   for J:=1 to  High(Auxiliar[I])-1 do //PARA PASAR ENTRE LOS ELEMENTOS
    eso.vp:= eso.vp + IntToStr(Auxiliar[I,J]) + ',';
   eso.Vp:= eso.vp + IntToStr(Auxiliar[I,High(Auxiliar[I])]);
   VActiva1.Store (I,Eso);
  end;{for I:=1 to high(Micodigos.ElemVp)}

 GuardarDatos;
 nT:=0;

 for I:=1 to Vactiva1.MaxVectorSize do //PARA MODIFICAR A LOS VP
  begin
   VActiva1.Recall(I,Eso);
   if (Eso.LVP>0)and(Eso.cApar=0) then inc(nT);
  end;{for I:=1 to Vactiva1.MaxVectorSize}

end;{TCodigos.OrdenVP}

{*****************************************************************}
{*****************************************************************}

procedure TCodigos.CrearMiDondeEstan; //sirve para decirme donde se encuentran los
grupos
var
 I,J,K,L,M,N, Aux,posicion:integer;
 A,B:word;
 AuxA : array of array of integer;
begin
//ORDENAMOS LOS GRUPOS PRIMERO EN EL ORDEN NATURAL
 for I:=0 to High(GruposA) do
  for J:=0 to High(GruposA[I]) do
   for K:=J to High(GruposA[I]) do
```

```
    if GruposA[I,J]>GruposA[I,K]
     then begin
      Aux:=GruposA[I,J];
      GruposA[I,J]:=GruposA[I,K];
      GruposA[I,K]:=Aux;
     end;{if GruposA[I,J]>GruposA[I,K] then}
//ORDENAMOS LOS GRUPOS ENTRE ELLOS MISMO
 SetLength(AuxA,1);
 for I:=0 to High(GruposA) do
  for J:=I+1 to High(GruposA) do
   if  GruposA[I,0]>GruposA[J,0]
    then begin
     for K:=0 to High(GruposA[I]) do
      begin
       SetLength(AuxA[0],K+1);
       AuxA[0,k]:=GruposA[I,k];  //ya copie al mas pequeño
      end;
     SetLength(GruposA[I],High(GruposA[J])+1); //cambio el tamaño del guardado  al del
otro
     // SetLength(GruposA[J],High(GruposA[I]));
     for K:=0 to High(GruposA[I]) do
      GruposA[I,k]:=GruposA[J,k];
     SetLength(GruposA[J],High(AuxA[0])+1);
     for K:=0 to High(GruposA[J]) do
      GruposA[J,k]:=AuxA[0,k];
    end;{if  GruposA[I,0]>GruposA[J,0] then}
//HASTA AQUI TENGO LOS GRUPOS YA ORDENADOS Y MODIFICADOS

//AHORA BUSCO A LOS GRUPOS AQUI
 SetLength(MiDondeGrupos,0);
 SetLength(MiDondeGrupos,VActiva1.MaxVectorSize + 1); //vector que me dira que
grupos contiene los lvp
 for I:=0 to High(GruposA) do //BUSCARE POR GRUPO EN LOS VP
  begin
   for J:=1 to High(ElemVp) do  //J revisar esta parte
    begin
     A:=0;
     if High(ElemVp[J])>1  //SI HAY MAS DE UN ELEMENTO ENTONCES BUSCA
      then begin
       For K:=0 to High(GruposA[I]) do //BUSCA SI HAY SE ENCUENTRA EL GRUPO
        For L:=0 to High(ElemVp[J]) do
         if GruposA[I,K]=ElemVp[J,L] then inc(A);
       if A=High(GruposA[I])+1
        then begin
         N:=High(MiDondeGrupos[J])+1;
         SetLength(MiDondeGrupos[J],N+1);
         MiDondeGrupos[J,N]:=I+1;
```

```
      end;{if A=High(GruposA[I])+1 then}
     end;{if High(ElemVp[J])>1 then}
    end;{for J:=1 to High(ElemVp)}
  end;{for I:=0 to High(GruposA)}          //AQUI YA ENCONTRE DONDE SE
ENCUENTRAN LOS GRUPOS
end;{TCodigos.CrearMiDondeEstan}


{*************************************************************}
{*************************************************************}
procedure TCodigos.MiOrden;
var
 eso, eso1, esoAux:TVActiva;
 VActiva2, VActiva3:AVector;
 I,j,k,l:integer;
 A:integer;
 NuevoOrden : array of integer;
 C320 : array of integer;
 Nuevo: array of array of integer;
begin
 SetLength(NuevoOrden,VActiva1.MaxVectorSize+1);
 VActiva2.Init (VActiva1.MaxVectorSize);
 VActiva3.Init (VActiva1.MaxVectorSize);
 A:=1;

 for I:=1 to VActiva1.MaxVectorSize do   //ME ACOMODA PRIMERO A LOS CEROS
  begin                           //DEL LVP
   VActiva1.Recall (I,eso);
   if eso.LVP = 0 then
    begin
     VActiva2.Store(A,eso);
     A:=A+1;
    end;{if eso.LVP = 0 then}
  end;{for I:=1 to VActiva1.MaxVectorSize}

 for I:=1 to A-2 do   //ME ACOMODA PRIMERO A LOS CEROS
  begin
   for J:=I+1 to A-1 do
    begin
     VActiva2.Recall (I,eso);                    //DEL LVP
     VActiva2.Recall (J,eso1);
     if (eso.cApar   > eso1.cApar ) then
      begin
       VActiva2.Store(I, eso1);//carga ambos datos
       VActiva2.Store(J, eso);
      end;{if (eso.cApar   > eso1.cApar ) then}
    end;{for J:=I+1 to A-1}
```

```
  end;{for I:=1 to A-2}


 for I:=1 to VActiva1.MaxVectorSize do // ACOMODA A TODOS LOS QUE SU
  begin                              //LVP Y CAPA <>0
   VActiva1.Recall (I,eso);
   if (eso.LVP <> 0) and (eso.cApar  <> 0) then
    begin
     VActiva2.Store(A,eso);
     A:=A+1;
    end;{if (eso.LVP <> 0) and (eso.cApar  <> 0) then}
  end;{for I:=1 to VActiva1.MaxVectorSize}

 for I:=1 to VActiva1.MaxVectorSize do
  begin
   VActiva1.Recall (I,eso);
   if (eso.LVP <> 0) and (eso.cApar  = 0) then    //ME ACOMODA ALOS CAPA=0
    begin
     VActiva2.Store(A,eso);
     A:=A+1;
    end;{if (eso.LVP <> 0) and (eso.cApar  = 0) then}
  end;{for I:=1 to VActiva1.MaxVectorSize}


 for I:=1 to VActiva1.MaxVectorSize do
  begin                  //GENERA UN NUEVO ARREGLO
   VActiva2.Recall  (I,eso);  //EL  CUAL  SERVIRA  PARA  SABER  COMO  SE
CAMBIARAN LOS VALORES DE
   NuevoOrden[eso.Index]:=I;  //ElemVp  PARA  DESPUES  SER  ALMACENADOS
NUEVAMENTE
  end;{for I:=1 to VActiva1.MaxVectorSize}

 VActiva1:=VActiva2;

 a:=MiCodigos.GuardarDatos; //aqui los pongo en su lugar nuevo al vp

 for I:=1 to high(Micodigos.ElemVp) do //AQUI CAMBIO DE VALOR A MI ELEMVP
SEGUN EL NUEVO ORDEN
  for J:=1 to  High(ElemVp[I]) do //PARA PASAR ENTRE LOS ELEMENTOS
   Elemvp[I,J]:= NuevoOrden[Elemvp[I,J]];

 for I:=1 to high(Micodigos.ElemVp) do //PARA MODIFICAR A LOS VP
  begin
   VActiva2.Recall(I,Eso);
   Eso.VP:='';
   for J:=1 to  High(ElemVp[I])-1 do //PARA PASAR ENTRE LOS ELEMENTOS
    eso.vp:= eso.vp + IntToStr(Elemvp[I,J]) + ',';
```

```
  eso.Vp:= eso.vp + IntToStr(Elemvp[I,High(ElemVp[I])]);
  VActiva2.Store (I,Eso);
 end;{for I:=1 to high(Micodigos.ElemVp)}

for I:=0 to high(Micodigos.GruposA ) do //PARA MODIFICAR A LOS GRUPOS ---->
 for J:=0 to  High(Micodigos.GruposA [I]) do
  Micodigos.GruposA  [I,J]:=  NuevoOrden[Micodigos.GruposA[I,J]];  //modifica los
numero en el orden los nuevos grupos


for I:=1 to VActiva2.MaxVectorSize do
 begin
  VActiva2.Recall (I,eso);
  Eso.Index :=I;
  Vactiva2.Store(I,Eso);
 end;

Vactiva1:=Vactiva2;

{*****************************************************}
//ME DA EL ORDEN SI HAY UNA ANTES
{*****************************************************}
for I:=1 to High(ElemVp) do
 NuevoOrden[I]:=0;

for I:=1 to High(ElemVp) do
 begin
  if NuevoOrden[I] = 0 then
   NuevoOrden[I]:=I;
  for J:=1 to High(ElemVp[I]) do
   if ElemVp[I,J]> I then
    if NuevoOrden[ElemVp[I,J]]= 0 then
     begin
      for K:=1 to High(ElemVp) do
       if NuevoOrden[K]>=I then NuevoOrden[K]:=K+1;
      NuevoOrden[ElemVp[I,J]]:=I;
     end;{if NuevoOrden[ElemVp[I,J]]= 0 then}
 end;{for I:=1 to High(ElemVp)}


for I:=1 to high(Micodigos.ElemVp) do //AQUI CAMBIO DE VALOR A MI ELEMVP
SEGUN EL NUEVO ORDEN
 for J:=1 to  High(ElemVp[I]) do //PARA PASAR ENTRE LOS ELEMENTOS
  Elemvp[I,J]:= NuevoOrden[Elemvp[I,J]];

for I:=1 to high(Micodigos.ElemVp) do //PARA MODIFICAR A LOS VP
 begin
```

```
  VActiva2.Recall(I,Eso);
  Eso.VP:='';
  for J:=1 to  High(ElemVp[I])-1 do //PARA PASAR ENTRE LOS ELEMENTOS
   eso.vp:= eso.vp + IntToStr(Elemvp[I,J]) + ',';
  eso.Vp:= eso.vp + IntToStr(Elemvp[I,High(ElemVp[I])]);
  VActiva2.Store (I,Eso);
 end;

 for I:=0 to high(Micodigos.GruposA ) do //PARA MODIFICAR A LOS GRUPOS ---->
  for J:=0 to  High(Micodigos.GruposA [I]) do
   Micodigos.GruposA [I,J]:= NuevoOrden[Micodigos.GruposA[I,J]]; //modifica los
numero en el orden los nuevos grupos


 for I:=1 to VActiva2.MaxVectorSize do
  begin
   VActiva2.Recall (NuevoOrden[I],eso);
   eso.index:=I;
   Vactiva3.Store(I,eso);
  end;

 for I:=1 to VActiva2.MaxVectorSize do
  begin
   VActiva3.Recall (I,eso);
   Vactiva1.Store(I,eso);
  end;

 Vactiva1:=Vactiva3;

 a:=MiCodigos.GuardarDatos; //aqui los pongo en su lugar nuevo al vp

end;{TCodigos.MiOrden}

{****************************************************************}
{****************************************************************}

procedure TCodigos.Ordenar;
var
 i,j,d,l, temp, n, k:integer;
 eso, eso1, eso2: TVActiva;
begin
 n:=VActiva1.MaxVectorSize;
 d:=1;
 while(d < n) do
  d :=d* 2;
 repeat
  d := d div 2;
```

```
  for I:=1 to n-d+1 do
   begin
    j:=i;
    l:=j+d;
    VActiva1.Recall(l, eso);//carga ambos datos
    VActiva1.Recall(j, eso1);
    while((j >= 1) and (eso.LVP < eso1.LVP)) do
     begin
      eso2 := eso;
      eso := eso1;
      eso1 := eso2;
      VActiva1.Store(l, eso);//carga ambos datos
      VActiva1.Store(j, eso1);
      l:=j;
      j:=j-d;
     end;{while((j >= 1) and (eso.LVP < eso1.LVP))}
   end;{for I:=1 to n-d+1}
 until d<=0;
end;{TCodigos.Ordenar}

{*************************************************************}
{*************************************************************}

procedure TCodigos.GuardaCM32;
var
 eso: TVActiva;
 i,j,k,l,m,n,a,z:Integer;
 compara:integer;
 compara2:integer;
 Orden:array of array of integer;
 GruposArreglados : array of array of integer;
 NuevoDondeEsta : array of array of integer;
begin
 compara:=0;
 Compara2:=0;
 for I:=1 to VActiva1.MaxVectorSize do       //PRIMERO PONGO EN TODOS UNOS Y
CEROS
  begin
   VActiva1.Recall(I, eso);
   if (eso.cApar = 3 )
    then eso.cM3:=1
    else eso.cM3:=0;
   VActiva1.Store(I, eso);
  end;{for I:=1 to VActiva1.MaxVectorSize}

 for I:=0 to High(GruposA)-1 do // ESTOS FOR SON PARA COMPARAR
  for J:=I+1 to High(GruposA) do // EL PRIMER GRUPO CON EL SEGUNDO
```

```
  begin
   if (High(GruposA[I]) <> High(GruposA[J])) then
    begin
     compara:=0;
     Compara2:=0;
     for K:=0 to High(GruposA[I]) do //para Verificar en los grupos
      for L:=0 to High(GruposA[J]) do
       if GruposA[I,K]=GruposA[J,L] then compara:=compara+1;
     //esto puede estar mal por si la cadena contiene n
     if compara-1= High(GruposA[I]) then    //SI ESTE GRUPO I ES SUBGRUPO DE
OTRO J
      begin
       for L:=0 to High(GruposA[I]) do //para Verificar en los grupos
        for M:=0 to High(GruposA[J]) do
         if GruposA[J,M]=GruposA[I,L] then GruposA[J,M]:=0;
       {for z:=0 to High(NuevoDondeEsta[J]) do
        begin
         if NuevoDondeEsta[J,compara2]<>0 then
         Compara2:=Compara2+1;
        end;
        SetLength(NuevoDondeEsta[J],compara2+1);
        NuevoDondeEsta[J,compara2]:=I+1;//PARA DECIRME QUE ESE ESTABA
COMPUESTO DE OTRO GRUPO
        compara2:=compara2+1;}
      end;{if compara-1= High(GruposA[I]) then }
     if compara-1= High(GruposA[J]) then //O SI J ES UN SUBGRUPO DE I
        for L:=0 to High(GruposA[I]) do //para Verificar en los grupos
         for M:=0 to High(GruposA[J]) do
          if GruposA[I,L]=GruposA[J,M] then GruposA[I,L]:=0;  //ELIMA A LOS QUE SON
SUBGRUPOS DEL ORIGINAL
    end;{if (High(GruposA[I]) <> High(GruposA[J])) then}
  end;{for J:=I+1 to High(GruposA)}    //AQUI ME DA LOS GRUPOS DEPURADOS AL
CIEN PORCIENTO
      //O MAS BIEN LE PONGO CEROS A LOS ELEMNTOS QUE SON SUBGRUPOS
DE LOS OTROS

 K:=1;
 SetLength(GruposArreglados,K);
 A:=0;

 for I:=0 to High(GruposA) do //AQUI LA VOLVEMOS A GUARDAR YA ORDENADA
  begin
   for J:=0 to High(GruposA[I]) do//PARA PASAR ENTRE LOS ELEMENTOS
    if GruposA[I,J]<>0 then      //Y YA PASARLO A OTRO
     begin
      A:=A+1;
      SetLength(GruposArreglados[k-1],A);
```

```
      GruposArreglados[K-1,A-1]:=GruposA[I,J];
     end;{if GruposA[I,J]<>0 then }
   if (High(GruposArreglados[K-1])>=1) then   //PARA ELIMINAR LOS REPETIDOS
    begin
     K:=K+1;
     SetLength(GruposArreglados,K);
    end;{if (High(GruposArreglados[K-1])>=1) then}
   A:=0;
  end;{for I:=0 to High(GruposA)}

//PARA VOLVER A PASARLOS YA BIEN ORDENADOS A GRUPOSA
 SetLength(GruposA,High(GruposArreglados));

 for I:=0 to High(GruposArreglados)-1 do //para dar los nuevos tamaños
  SetLength(GruposA[I],High(GruposArreglados[I])+1);
 for I:=0 to High(GruposArreglados) do //FOR S PARA VOLVELOS A PASAR
  for J:=0 to High(GruposArreglados[I]) do
   GruposA[I,J]:=GruposArreglados[I,J]; //YA COMPLETAMENTE DEPURADOS Y NO
REPETIDOS

 for I :=0   to High(GruposA) do          //LES PONE 2 A LOS QUE EXISTEN EN LOS
GRUPOS Y LO GUARDO
  for J := 0 to High(Micodigos.GruposA[I]) do
   begin
    VActiva1.Recall(Micodigos.GruposA[I,J], eso);
    eso.cM3:=2;
    VActiva1.Store(Micodigos.GruposA[I,J], eso);
   end;{for J := 0 to High(Micodigos.GruposA[I])}

end;{TCodigos.GuardaCM32}

{***************************************************************}
{***************************************************************}

procedure TCodigos.Cm32;
var
 I, J:Integer;
 K, L:integer;
 M:Integer;
 A,N:Integer;
 posicion:integer;
 ColaCm32:Queue;
 Grupos : array of array of integer;
 DondeGrupos:array of array of integer;
 OrdenGrupos:array of integer;
 eso: TVActiva;
```

```
begin
 ColaCm32.init(SizeOf(Integer), 10);
 A:=0; //PARA SABER EN QUE ELEMENTO DEL ARREGLO GRUPO SE PONE
 posicion:=0;
 SetLength(Grupos,1);
 SetLength(DondeGrupos,0);
 SetLength(DondeGrupos,VActiva1.MaxVectorSize+1);

 for I:=1 to VActiva1.MaxVectorSize-1 do            //ESTOS DOS CICLOS ME HACEN
DAR LA VUELTA
  for J:=I+1 to VActiva1.MaxVectorSize do  //PARECIDA A LA DE LA BURBUJA
   begin   //J E I ME DICEN EN QUE COLUMNA ESTOY
    if(High(ElemVp[I])<> High(ElemVp[J]))then   //SI LA LONGUITUD DE VECTOR ES
DIFERENTE
     for K:=1 to   High(ElemVp[I]) do             //ESTOS DOS FOR ME EXAMINAN A
TODOS
      for L:=1 to High(ElemVp[J]) do   //ELEMENTOS DE CADA UNO
       if (ElemVp[I,K]=ElemVp[J,L])
        then ColaCm32.EnQueue(ElemVp[I,K], SizeOf(Integer));
    if(ColaCm32.Items_In>=2) then
     begin
      SetLength(Grupos[A],ColaCm32.Items_In);
      for M:=0 to ColaCm32.Items_In-1 do
       ColaCm32.DeQueue(Grupos[A,M],SizeOf(Integer) );   //Para Guardar los grupos
      A:=A+1;
      Posicion:=0;
      for N:=0 to High(DondeGrupos[I]) do  //para decir donde se encentran los grupos
       if DondeGrupos[I,Posicion]<>0 then posicion:=posicion+1;
      SetLength(DondeGrupos[I],posicion+1);
      DondeGrupos[I,Posicion]:=A;
      Posicion:=0;
      for N:=0 to High(DondeGrupos[J]) do //para decir donde se encuentran los grupos
       if DondeGrupos[J,Posicion]<>0 then posicion:=posicion+1;
      SetLength(DondeGrupos[J],posicion+1);
      DondeGrupos[J,Posicion]:=A;
      SetLength(Grupos,A+1);
     end;{if(ColaCm32.Items_In>=2) then}
    ColaCm32.Clear;
    Posicion:=0;
   end;{for J:=I+1 to VActiva1.MaxVectorSize} //FIN DEL FOR I y J

 SetLength(OrdenGrupos,High(Grupos));

 for I:=0 to High(OrdenGrupos) do
  OrdenGrupos[I]:=I+1;

 for I:=0 to High(Grupos)-2 do     //PARA OBTENER GRUPOS NO REPETIDOS
```

```
  for J:=I+1 to High(Grupos)-1 do
   begin
    if (High(Grupos[I])=High(Grupos[J])) then
     begin
      A:=0;
      if (High(Grupos[I])>0) then
       for K:=0 to High(Grupos[I])do
        for L:=0 to High(Grupos[I]) do
         if Grupos[I,K]=Grupos[J,L] then A:=A+1;
      if (A-1=High(Grupos[I])) then
       begin
        SetLength(Grupos[J],1); //AQUI ME PONE EN cero A LOS REPETIDOS
        OrdenGrupos[J]:=I+1;
       end;{if (A-1=High(Grupos[I])) then}
     end;{if (High(Grupos[I])=High(Grupos[J]))}  //if
   end;{for J:=I+1 to High(Grupos)-1} //FIN FOR J E I

 {for I:=0 to High(DondeGrupos)-1 do          //PARA OBTENER DONDE SE ENCUENTRAN LOS GRUPOS
       for J:=0 to High(DondeGrupos[I]) do
       DondeGrupos[I,J]:=OrdenGrupos[DondeGrupos[I,J]-1];  }

 posicion:=High(DondeGrupos[0]);
 A:=0;

 for I:=0 to High(Grupos)do   //PARA AHORAR MEMORIA
  begin
   if (High(Grupos[I])>0) then //Y COPIARLOS EN GRUPOS A
    begin
     A:=A+1;
     SetLength(GruposA,A);
     SetLength(GruposA[A-1],High(Grupos[I])+1);
     for J:=0 to High(Grupos[I]) do
      GruposA[A-1,J]:=Grupos[I,J];
    end;{if (High(Grupos[I])>0) then }

   {if I< High(Grupos) then
   OrdenGrupos[I]:=High(GruposA)+1;}
  end;{for I:=0 to High(Grupos)do}

 {for I:=0 to High(DondeGrupos)-1 do          //PARA OBTENER DONDE SE ENCUENTRAN LOS GRUPOS
       for J:=0 to High(DondeGrupos[I]) do
       DondeGrupos[I,J]:=OrdenGrupos[DondeGrupos[I,J]-1];

 SetLength(MiDondeGrupos,High(DondeGrupos)+1);
```

```
 for I:=0 to High(DondeGrupos) do
  begin
 SetLength(MiDondeGrupos[I],High(DondeGrupos[I])+1);
       for J:=0 to High(DondeGrupos[I]) do
       begin
       MiDondeGrupos[I,J]:=DondeGrupos[I,J];     //DE AQUI ME DICE QUE GRUPOS
CONTIENE Y EN DONDE ESTAN
       end;                              //REPETIDOS Y TODO 1,2,3,3,4
  end;//}

end;{TCodigos.Cm32}

{****************************************************************}
{****************************************************************}

procedure TCodigos.CodAparicion;
var
 I:integer;
 J:integer;
 L:integer;
 K:integer;
 eso: TVActiva;
begin
 SetLength(CodApa,VActiva1.MaxVectorSize+2);
 for I:=1 to VActiva1.MaxVectorSize  do
  SetLength(CodApa[I],3);
 for I:=1 to VActiva1.MaxVectorSize   do     //ESTE FOR ES PARA LLENAR LA
TABLITA DE CEROS
  for J:=1 to 2  do
   CodApa[I,J]:=0;

 for I := 1 to VActiva1.MaxVectorSize do
  for J := 1 to High(ElemVp[I]) do     //HASTA EL NUMERO DE ELEMNTOS
   begin                              //ESTOS FOR ME LLENAN LA TABLA QUE LES
EXPLIQUE
    if (ElemVp[I,J]<>0) then
     begin
      if (High(ElemVp[I])=1) then CodApa[ElemVp[I,J], 1]:=1; //QUE SI VIENE SOLO
ESE ELEMENTO LE PONENE UN 1
      if (High(ElemVp[I])>1) then CodApa[ElemVp[I,J], 2]:=1; //QUE SI NO VIENE SOLO
OSEA ACOMPAÑADO LE PONE UN 1
     end;{if (ElemVp[I,J]<>0)}
   end;{for J := 1 to High(ElemVp[I])}

 //SE MANEJA POR FILA COLUMNA DONDE LA FILA SERA EL INDICE

 for K:=1 to VActiva1.MaxVectorSize do
```

```
 for L:=1 to High(CodApa[K]) do
  begin
   VActiva1.Recall(K, eso);
   //SOLO              ACOMPAÑADO
   if (CodApa[K, 1]=0) and (CodApa[K, 2]=0)
    then eso.cApar:=0  //NO APARECE
    else if (CodApa[K, 1]=1) and (CodApa[K, 2]=0)
     then eso.cApar:=1   //APARECE SOLO
     else if (CodApa[K, 1]=1) and (CodApa[K, 2]=1)
      then eso.cApar:=2 //APARECE SOLO Y ACOMPAÑADO
      else if (CodApa[K, 1]=0) and (CodApa[K, 2]=1) then eso.cApar:=3;  //APARECE
ACOMPAÑADO
   VActiva1.store(K, eso);
  end;{for L:=1 to High(CodApa[K])}
end;{TCodigos.CodAparicion}

{**************************************************************}
{**************************************************************}

Function TCodigos.a(X:integer):integer;
var
eso1: TVActiva;
b:string;
begin
VActiva1.Recall(X, eso1);
b:=Eso1.vp;
result:=length(eso1.vp);
end;{TCodigos.a}

{**************************************************************}
{**************************************************************}

function TCodigos.GuardarDatos:integer;     {ME ALMACENA LOS DATOS EN EL
ARREGLO DE LA CLASE}
var                          {SI UNO SOBREPASA EL LIMITE RETORNA ERROR}
 eso1: TVActiva;
 k:integer;
 I, J:integer;
 L, M:integer;
 StrCola:string[4];
 IntCola:integer;
 colaz:Queue;
begin
 SetLength(ElemVp,0);
 SetLength(ElemVp,(VActiva1.MaxVectorSize+1)); //GENERO UN ARREGLO CON
MaxVectorSize DE FILAS
 colaz.init(SizeOf(integer),11);
```

```
 for k:=1 to VActiva1.MaxVectorSize do //PARA PASAR POR TODOS LOS
ELEMENTOS DE VACTIVA1
  begin
   VActiva1.Recall(k, eso1);
   colaz.clear;
   J:=1;
   for I:=1 to length(eso1.VP) do
    begin
     if (eso1.VP[I] in ['0'..'9'] )
      then
       begin
        StrCola[J]:=eso1.VP[I];
        J:=J+1;
       end {if (eso1.VP[I] in ['0'..'9'] ) then}
      else if (eso1.VP[I]=',') then
       begin
        StrCola[0]:=chr(J-1);
        IntCola:=StrToInt(StrCola);
        IntCola:=StrToInt(StrCola);
        if (intCola>VActiva1.MaxVectorSize) then
         begin
          result:=1;
          exit;
         end;{if (intCola>VActiva1.MaxVectorSize) then}
        if (IntCola>0) then Colaz.EnQueue (IntCola, SizeOf(integer));
        J:=1;
       end;{if (eso1.VP[I] in ['0'..'9'] ) else}
    end;{for I:=1 to length(eso1.VP)} //FIN DEL FOR I
   StrCola[0]:=chr(J-1);
   IntCola:=StrToInt(StrCola);
   IntCola:=StrToInt(StrCola);
   if (intCola>VActiva1.MaxVectorSize) then
    begin
     result:=1;
     exit;
    end;{if (intCola>VActiva1.MaxVectorSize) then}
   if (IntCola>0) then Colaz.EnQueue (IntCola, SizeOf(integer));
   J:=1;
   Eso1.LVP:=Colaz.Items_In;
   VActiva1.Store(k, eso1);

   if (Colaz.Items_In>0)
    then
     begin
      SetLength(ElemVp[K],Colaz.Items_In+1); //QUE TENDRAN ESA FILA
      M:=Colaz.Items_In;                      //Colaz.Items_In ELEMENTOS
      for L:=1 to M do
```

```
      colaz.DeQueue(ElemVp[K, L],SizeOf(Integer) );
     end {if (Colaz.Items_In>0) then}
    else
     begin
      SetLength(ElemVp[K],2);
      ElemVp[K,1]:=0;
     end;{if (Colaz.Items_In>0) else}
  end;{for k:=1 to VActiva1.MaxVectorSize} //FIN FOR K
end;{TCodigos.GuardarDatos}

{****************************************************************}
{****************************************************************}

procedure TCodigos.VerifComas;  //PROCEDIMIENTO QUE MODIFICA, DEPURA Y
GUARDAR LOS ELEMENTOS DE
var                              //EL OBJETO VACTIVAL1
 eso: TVActiva;
 J, I, k, L, Estado:Integer;
 Dato, Arreglada:String[30];
begin
 for k:=1 to VActiva1.MaxVectorSize do //PARA PASAR POR TODOS LOS
ELEMENTOS DE VACTIVA1
  begin
   VActiva1.Recall(k, eso);//CARGO LOS ELEMENTOS DEL INDICE K EN "ESO
   J:=1;
   Estado:=1;     //0 el ultimo fue numero
                  //1 El Ultimo fue una coma
   Dato:=Eso.VP;  //LO PASO POR QUE NECESITO A LA CADENA CON INDICES
   L:=Length(Eso.vp);
   if (eso.VP='') then    //QUE SI NO ALMACENO UN DATO
    begin            //ES EQUIVALENTE A CERO
     eso.VP:='0';
     VActiva1.Store(k, eso);  //Y LO GUARDA YA ARREGLADO
     EXIT;
    end;
   if (length(Dato)<>0) then   //SI EL TAMAÑO DEL VP DE ESO ES MAYOR A CERO
    begin
     for I:=1 to length(Dato) do
      begin
       if (Dato[I] in ['0'..'9'] )
        then
         begin
          Arreglada[J]:=Dato[I];
          J:=J+1;
          Estado:=0;
         end {if (length(Dato)<>0) then}
        else if (Dato[I]=',') and (Estado=0) then
```

```
        begin
         Arreglada[J]:=Dato[I];
         J:=J+1;
         Estado:=1;
        end;{if (length(Dato)<>0) else if (Dato[I]=',') and (Estado=0) then}
      end;{for I:=1 to length(Dato} //FIN DEL FOR
     if (Arreglada[j-1]=',') then J:=J-1;   //SI EL ULTIMO ES COMA ME LO QUITA
     Arreglada[0]:=Chr(j-1); //ME DA LA LONGUITUD DE "ARREGLADA", ¡YA LA ARREGLO!
    end;{if (length(Dato)<>0) then} //FIN DEL IF GRANDE
   Eso.Vp[0]:=Chr(j-1);
   Eso.VP:=Arreglada;//MODIFICA EL VALOR DE ESO
   VActiva1.Store(k, eso);  //Y LO GUARDA YA ARREGLADO
  end;{for k:=1 to VActiva1.MaxVectorSize}
end;{TCodigos.VerifComas}

procedure TCodigos.borrar;
begin
 setLength(ElemVp,0);
 setLength(CodApa,0);
 setLength(GruposA,0);
 setLength(MiDondeGrupos,0);
 setLength(CR1,0);
end;

end.
```

```
unit UCPM;

interface
uses SysUtils,UDatos,UMain,URed;

procedure CargaTabla;
procedure Cpm;
var IndiceDos:array of word;

implementation

procedure Proximo;
var
 k,ki,nk:integer;
 A1,Ap:TRVcpm;
 flTp:boolean;
 maxTPx:real;{utilizada ara determinar el iPx de las actividades subsecuentes}
begin
 k:=0;
 DurPro:=0;
 {maxTPx:=0;}
 repeat {procesa las iniciales}
  inc(k);
  Vcpm.Recall(k,A1);
  if A1.i=1
   then begin
    A1.iPx:=0; {inicio próximo de la actividad}
    A1.tPx:=A1.Dur; {terminación próxima de la actividad}
    Vcpm.Store(A1.Index,A1);
   end;
 until A1.i>1;
 ki:=1;
 repeat
  inc(ki);
  flTp:=False;
  k:=0;
  repeat  {procesa todas las intermedias y las finales}
   inc(k);
   Vcpm.Recall(k,A1);
   if A1.j=ki
    then begin
     flTp:=True;
     maxTPx:=A1.tPx;
     nk:=k+1;
     repeat  {se utiliza para calcular maxTPx entre las acts. con misma j}
      Vcpm.Recall(nk,Ap);
      if Ap.j=ki
```

```
       then if Ap.tPx>maxTPx then maxTPx:=Ap.tPx;
      inc(nk);
     until nk>Vcpm.VSize;
     nk:=1;
     repeat   {determina los tiempos de inicio y terminación, próximo}
      Vcpm.Recall(nk,Ap);
      if Ap.i=ki
       then begin
        Ap.iPx:=maxTPx;
        Ap.tPx:=maxTPx+Ap.Dur;
        Vcpm.Store(Ap.Index,Ap);
       end;
      inc(nk);
     until nk>Vcpm.VSize;
    end;
  until (k=Vcpm.VSize)or flTp;
  if maxTPx>DurPro then DurPro:=maxTPx;
 until ki=JMax; {verificar que JMax sea el valor más alto para las j's}
end;{Proximo}

procedure Lejano;
var
 k,kj,nk:integer;
 A1,Ap:TRVcpm;
 flTL:boolean;
 miniLx:real;{integer;}
begin
 k:=Vcpm.VSize+1;
 repeat //se procesan las actividades terminales primero
  dec(k);
  Vcpm.Recall(k,A1);
  if A1.j=JMax
   then begin
    A1.tLx:=DurPro;
    A1.iLx:=A1.tLx-A1.Dur;
    Vcpm.Store(A1.Index,A1);
   end;
 until k<=1;
 kj:=JMax;
 repeat
  dec(kj);
  flTL:=False;
  k:=Vcpm.VSize+1;
  repeat
   dec(k);
   Vcpm.Recall(k,A1);
   if A1.i=kj
```

```
    then begin
     flTL:=True;
     miniLx:=A1.iLx;
     nk:=k-1;
     repeat
      Vcpm.Recall(nk,Ap);
      if Ap.i=kj
       then if Ap.iLx<miniLx then miniLx:=Ap.iLx;
      dec(nk);
     until nk<=1;
     nk:=Vcpm.VSize-1;
     repeat
      Vcpm.Recall(nk,Ap);
      if Ap.j=kj
       then begin
        Ap.tLx:=miniLx;
        Ap.iLx:=miniLx-Ap.Dur;
        Vcpm.Store(Ap.Index,Ap);
       end;
      dec(nk);
     until nk<1;
    end;
  until (k=1)or flTL;
 until kj=1;
end;{Lejano}

procedure Holguras;
var
 k,nk:integer;
 A1,Ap:TRVcpm;
 rRC:TRC;
 rEstima:TEstima;
 flOk:boolean;
begin
 k:=0;
 qHol.Init(SizeOf(rRC),1000);
 repeat
  inc(k);
  Vcpm.Recall(k,A1);
  A1.hTotal:=A1.iLx-A1.iPx;  //holgura total de la actividad
  if A1.hTotal=0  //selecciona las actividades sin holgura para la ruta crítica
   then begin
    SetLength(IndiceDos, High(IndiceDos)+2);
    IndiceDos[High(IndiceDos)]:=A1.Index;
    rRC.Index:=A1.Index;
    rRC.i:=A1.i;
    rRC.j:=A1.j;
```

```
    qHol.EnQueue(rRC,SizeOf(rRC));
    VEstima.Recall(A1.Index,rEstima);
    Sigm:=Sigm+rEstima.Varianza;
   end;
  nk:=A1.Index;
  A1.hLib:=0;
  flOk:=False;
  repeat
   inc(nk);
   Vcpm.Recall(nk,Ap);
   if Vcpm.VectorOpStatus and (A1.j=Ap.i)
    then begin
     A1.hLib:=Ap.iPx-A1.tPx; //calcula la hilgura libre
     flOk:=True;
    end;
  until flOk or(A1.hLib>0)or(not(Vcpm.VectorOpStatus));
  if not Vcpm.VectorOpStatus then A1.hLib:=DurPro-A1.tPx;
  A1.hInter:=A1.hTotal-A1.hLib; //calcula la holgura de interferencia
  Vcpm.Store(A1.Index,A1);
 until k=Vcpm.VSize;
end;{Holguras}

procedure CargaTabla;
var
 i,nk,k:integer;
 aFuente:TVActiva;
 aDestino:TRVcpm;
 sNk:shortString;
 rFic:TRFic;
 rEstima:TEstima;
begin
 FicVector.Init(ColaFic.Items_In);
 for i:=1 to ColaFic.Items_In do begin
  ColaFic.DeQueue(rFic,SizeOf(rFic));
  FicVector.Store(i,rFic);
 end;{for i..}
 nk:=FicVector.VSize;
 Vcpm.Init(VActiva1.VSize+nk);
 i:=0;
 repeat    //carga las iniciales en Vcpm
  inc(i);
  VActiva1.Recall(i,aFuente);
  VEstima.Recall(i,rEstima);
  if aFuente.LVP=0
   then begin
    aDestino.Index:=aFuente.Index;
    aDestino.i:=aFuente.i;
```

```
    aDestino.j:=aFuente.j;
    aDestino.Dur:=rEstima.DurMe;
    aDestino.iPx:=0;
    aDestino.tPx:=0;
    aDestino.iLx:=0;
    aDestino.tLx:=0;
    aDestino.hTotal:=0;
    aDestino.hLib:=0;
    aDestino.hInter:=0;
    Vcpm.Store(aDestino.Index,aDestino);
   end;
 until aFuente.LVP>0;
 for k:=1 to nk do begin   //carga las ficticias en Vcpm
  FicVector.Recall(k,rFic);
  aDestino.i:=rFic.i;
  aDestino.j:=rFic.j;
  aDestino.Dur:=0;
  aDestino.Index:=i;
  aDestino.iPx:=0;
  aDestino.tPx:=0;
  aDestino.iLx:=0;
  aDestino.tLx:=0;
  aDestino.hTotal:=0;
  aDestino.hLib:=0;
  aDestino.hInter:=0;
  Vcpm.Store(aDestino.Index,aDestino);
  inc(i);
 end;{for k...}
 repeat  //carga las intermedias y finales en Vcpm
  VActiva1.Recall(i-nk,aFuente);
  aDestino.Index:=aFuente.Index+nk;
  aDestino.i:=aFuente.i;
  aDestino.j:=aFuente.j;
  VEstima.Recall(i-nk,rEstima);
  aDestino.Dur:=rEstima.DurMe;{StrToFloat(sNK);}
  aDestino.iPx:=0;
  aDestino.tPx:=0;
  aDestino.iLx:=0;
  aDestino.tLx:=0;
  aDestino.hTotal:=0;
  aDestino.hLib:=0;
  aDestino.hInter:=0;
  if VActiva1.VectorOpStatus
   then Vcpm.Store(aDestino.Index,aDestino);
  inc(i);
 until not VActiva1.VectorOpStatus;
end;{CargaTabla}
```

```
procedure Cpm;
begin
 CargaTabla;
 Proximo;
 Lejano;
 Holguras;
end;

end.
```

```
unit Upert;

interface

uses SysUtils,UDatos,UMain,UValz;

 procedure cPert(cParm:TcParm);
implementation
procedure cPert(cParm:TcParm);
begin
 case StrToInt(cParm[1]) of
  1:CalcAreaZ(StrToFloat(cParm[2]),StrToInt(cParm[3]));
  2:CalcAreaZZ(StrToFloat(cParm[2]),StrToFloat(cParm[3]),StrToInt(cParm[4]));
  3:CalcAreaZZ(StrToFloat(cParm[2]),StrToFloat(cParm[3]),StrToInt(cParm[4]));
  4:CalcAreaZZ(StrToFloat(cParm[2]),StrToFloat(cParm[3]),StrToInt(cParm[4]));

5:CalZIzq(StrToFloat(cParm[2]),StrToFloat(cParm[3]),StrToFloat(cParm[4]),StrToInt(cPar
m[5]));
 end;{case}
end;{cPert(cParm:TcParm)}

end.
```

```
unit UValz;

interface

uses SysUtils;

 procedure CalZIzq(Pro,Mu,Sigma:real;Op:integer);
 procedure CalcAreaZ(TP:real;Op:integer);
 procedure CalcAreaZZ(TP1,TP2:real;Op:integer);

implementation

uses UMain,UDatos;
var
  i,j,M:integer;
  N :integer;//número de subintervalos en que se divide el intervalo de integración
  A,    //límite inferior del intervalo de integración
  B,    //límite superior del intervalo de integración
  Rich, //valor de la integral después de la interpolación de Richardson
  H,    //amplitud constante de los subintervalos
  X :real; //variable independiente de la función
  Area  :array[1..2]of real; //arreglo de áreas: (1)N subintervalos, (2)2N subintervalos
  K1,K2:longInt;
  nDatos:integer;
  Dato1,Dato2:real;
  Izq:boolean;
  ArZ:real;

function Y(x:real):real;
begin
 Y:=Exp(-(x*x)/2)/Sqrt(2*pi);
end;{Y}

function AreaBC(A,B:real;N:integer):real;
//Calcula el área bajo la curva con la fórmula trapecial (Luthe pp. 170-181)
begin
 for i:=1 to 2 do begin
  H:=(B-A)/N;
  X:=A;
  Area[i]:=0; //inicializa la i-ésima sumatoria
  M:=N-1; //número de subintervalos para el cálculo
  for j:=1 to M do begin //ciclo que obtiene la sumatoria de las áreas de los subintervalos
   X:=X+H;
   Area[i]:=Area[i]+Y(X);
  end;{for j}
  Area[i]:=H*(Y(A)+Y(B)+2*Area[i])/2; //fórmula trapecial
  N:=2*N;
```

```
 end;{for i}
 Rich:=(4*Area[2]-Area[1])/3; //interpolación de Richardson entre las dos sumatorias: N y 2N subintervalos
 AreaBC:=Rich;
end;{AreaBC}

function Kalcula(AreaTar:real):real;
//Realiza la exploración para determinar la Z buscada a partir de un área determinada
var
 H:real;
 Izq:boolean;
 f1,f2, //factores de aproximación en la exploración
 ArZ:real;
 b1,b2, //límites del intervalo de exploración
 Area1,Area2:real;
 ZNeg:boolean;
 k:integer;
begin
   K1:=0;
   K2:=0;
   k:=0;{contador}
   ZNeg:=False;
   A:=-4;  {Valor inicial para la variable aleatoria en el límite inferior del rango explorador}
   B:=-3.9; {Valor inicial para la variable aleatoria en el límite superior del rango explorador}
   N:=16;  {Número de intervalos en dividirá el rango explorador}
   H:=(B-A)/N; {ancho del intervalo para el rango explorador}
   if AreaTar=0.5 then Area1:=AreaTar  {variable aleatoria en el centro de la distribución}
   else begin
    if AreaTar<0.5
     then begin   {utilizado para realizar los cálculos en el lado positivo simétrico de la curva}
      AreaTar:=1-AreaTar;
      ZNeg:=True;
     end;
    b2:=B; {b2:=b1; {valor inicial del límite superior del primer intervalo}
    Area2:=AreaBC(A,b2,N); {área del límite superior del primer intervalo}
    repeat   {cálculo de áreas sucesivas hasta revasar el área de la variable aleatoria desconocida}
     inc(K1);
     inc(k);{espía}
     b1:=b2;
     Area1:=Area2; {preserva el área de la cota inferior del intervalo}
     b2:=b2+H;  {avanza al siguiente intervalo}
     Area2:=AreaBC(A,b2,N); {calcula el área del nuevo limite superior del intervalo}
    until (Area2>=AreaTar);//Se detiene el proceso cuando el intervalo envuelve a la variable aleatoria buscada
```

```
   {or(k=100000); {*** si no converge el método, MaxInt=2147483647  se utilizó con
algunos ejemplos de prueba}
   f1:=(AreaTar-Area1)/AreaTar; {primer factor de ajuste para acercar el límite b1 a bx}
   K1:=k;
   f2:=(Area2-AreaTar)/AreaTar; {segundo factor de ajuste para acercar el límite b2 a bx}
   while b1<b2 do begin  {ciclo utilizado para cruzar los valores exploradores b1 y b2}
    inc(K2);
    if b1>=0
      then b1:=b1*(1+f1)
      else b1:=b1*(1-f1);
    if b2>=0
     then b2:=b2*(1-f2)
     else b2:=b1*(1+f2);
   end;{while}
   b1:=(b1+b2)/2;
   Area1:=AreaBC(A,b1,N);
  end;{else del if AreaTar=0.5}
 {Area2:=AreaTar-Area1; instrucción aparentemente inútil}
 if ZNeg then b1:=-b1; {para valores simétricos}
 Kalcula:=b1; {valor buscado}
end;

procedure CalcAreaZ(TP:real;Op:integer);
var
 A:real;
 N:integer;
begin
 A:=-4;
 N:=16;
 Dato1:=TP;
 ArZ:=AreaBC(A,Dato1,N);

 case Op of
  1:begin
    fmMain.lbProba.Caption:='La probabilidad que se relice en '+sTP+' o menos es =
'+Format('%f %%',[ArZ*100]);{FloatToStr(ArZ*100)+'  %';}
    fmMain.lbProba.Visible:=True;
    if fmMain.Vercomplemento1.Checked then
     fmMain.lbProbaCom.Caption:='La probabilidad que no se relice en '+sTP+' es =
'+Format('%f %%',[(1-ArZ)*100]);{FloatToStr((1-ArZ)*100)+'  %';}
   end;
  2:begin
    fmMain.lbProbaCom.Caption:='La probabilidad que no se relice es = '+Format('%f
%%',[(1-ArZ)*100]);{FloatToStr((1-ArZ)*100)+'  %';}
    fmMain.lbProbaCom.Visible:=True;
    fmMain.lbProba.Visible:=False;
   end;
```

```
 end;{case}
 {fmMain.lbProba.Visible:=True;
 fmMain.lbProbaCom.Visible:=True;}
end;{CalcAreaZ}

procedure CalcAreaZZ(TP1,TP2:real;Op:integer);
var
 A,TPtempo:real;
 N:integer;
begin
 N:=16;
 if TP2<=TP1 then begin
   TPtempo:=TP1;
   TP1:=TP2;
   TP2:=TPtempo;
  end
  else begin
   ArZ:=AreaBC(TP1,TP2,N);
   case Op of
    1:begin
      fmMain.lbProba.Caption:='La probabilidad que se relice entre TP1 y TP2 es =
'+Format('%f %%',[ArZ*100]);{FloatToStr(ArZ*100)+'  %';}
      fmMain.lbProba.Visible:=True;
      fmMain.lbProbaCom.Visible:=False;
      {fmMain.lbProbaCom.Caption:='La probabilidad que no se relice en T1-T2 es =
'+Format('%f %%',[(1-ArZ)*100]);{FloatToStr((1-ArZ)*100)+'  %';}
     end;
    2:begin
      fmMain.lbProba.Caption:='La probabilidad que se relice alrededor de T es =
'+Format('%f %%',[ArZ*100]);{FloatToStr(ArZ*100)+'  %';}
      {fmMain.lbProbaCom.Caption:='La probabilidad que no se relice alrededor de T es =
'+Format('%f %%',[(1-ArZ)*100]);{FloatToStr((1-ArZ)*100)+'  %';}
      fmMain.lbProbaCom.Visible:=False;
     end;
    3:begin
      fmMain.lbProba.Caption:='La probabilidad que se relice exactamente en T es =
'+Format('%f %%',[ArZ*100]);{FloatToStr(ArZ*100)+'  %';}
      {fmMain.lbProbaCom.Caption:='La probabilidad que no se relice exactamente en T es
= '+Format('%f %%',[(1-ArZ)*100]);{FloatToStr((1-ArZ)*100)+'  %';}
     end;
    4:begin
      fmMain.lbProbaCom.Caption:='La probabilidad que no se relice entre TP1 y TP2 es =
'+Format('%f %%',[(1-ArZ)*100]);{FloatToStr((1-ArZ)*100)+'  %';}
      fmMain.lbProbaCom.Visible:=True;
      fmMain.lbProba.Visible:=False;
     end;
   end;{case}
```

```
   {fmMain.lbProbaCom.Visible:=True;}
  end;{if TP2<=TP1 else}
end;{CalcAreaZZ}

procedure CalZIzq(Pro,Mu,Sigma:real;Op:integer);
var
 rZ,TP:real;
begin
 rZ:=Kalcula(Pro);
 TP:=Mu+rZ*Sigma;
 fmMain.lbProba.Caption:='El tiempo necesario para una brobabilidad de '+FloatToStr(Pro*100)+'% es '+ FloatToStr(TP);
 fmMain.lbProba.Visible:=True;
end;{CalZIzq}
end.
```

```
unit UDatos;

interface

const
 maxGpos=10;


Type

 TRIndex=record
         Index:word;
         i,j:word;
         LenVP:word;
         Ok:boolean;
        end;

 TEstima= record
          Index:word;
          tOp,
          tMe,
          tPe,
          DurMe,
          Varianza:real;
         end;

 OneInt= ARRAY [1..1] OF word;
 OneIntPtr = ^OneInt;

 WVector = OBJECT
   MaxVectorSize,
   VSize : WORD;
   VecPtr : OneIntPtr;
   VectorOpStatus : BOOLEAN;
   CONSTRUCTOR Init(MaxElem : WORD { input });
   DESTRUCTOR  Done;
   FUNCTION CurrentSize : WORD;
   PROCEDURE Store ( Index  : WORD; { input  }
                   X      : word  { input  });
   PROCEDURE Recall( Index  : WORD; { input  }
               VAR  X      : word  { output });
   PROCEDURE Fill(   FillValue : word; { input }
                   NumElem   : WORD  { input });
   PROCEDURE AddScalar(Scalar : word { input });
   PROCEDURE MultScalar(Scalar : word { input });
   PROCEDURE AddVector(VectB : WVector { input });
   FUNCTION MultVector(VectB : WVector { input }) : word;
```

```
END;

OneEstima= ARRAY [1..1] OF TEstima;
OneEstimaPtr = ^OneEstima;

TVEstima = OBJECT
  MaxVectorSize,
  VSize : WORD;
  VecPtr : OneEstimaPtr;
  VectorOpStatus : BOOLEAN;
  CONSTRUCTOR Init(MaxElem : WORD { input });
  DESTRUCTOR  Done;
  FUNCTION CurrentSize : WORD;
  PROCEDURE Store ( Index  : WORD; { input  }
                X      : TEstima  { input  });
  PROCEDURE Recall( Index  : WORD; { input  }
            VAR  X      : TEstima  { output });
  PROCEDURE Fill(   FillValue : TEstima; { input }
                NumElem   : WORD  { input });
END;


    StatusType    = Object
                  {REPRESENTATION}
                   OverFlow,
                   UnderFlow,
                   Object_MisMatch,
                   Dynamic_Memory_Exhausted,
                   General_Failure          : BOOLEAN;

                  {SPECIFICATION}
                   PROCEDURE Reset;
                 END;

    StackNodePtr  = ^StackNode;

    StackNode     =
       Object
        {REPRESENTATION}
         Info       : POINTER;
         Next       : StackNodePtr;

        {SPECIFICATION}
         PROCEDURE Init (  VAR Captured_Info; {untyped}
                           Info_Size : WORD;
                           Next_Node : StackNodePtr);
```

```
    PROCEDURE Yield_Info ( VAR Yielded_Info; {untyped}
                           Info_Size : WORD     );

    PROCEDURE Clear (Info_Size : WORD);
   END;

 Stack        =
   Object
    {REPRESENTATION}
     Top       : StackNodePtr;
     Size      : WORD;
     MaxSize   : WORD;
     InfoSize  : WORD;
     Last_Op_Ok : BOOLEAN;
     Status    : StatusType;

    {SPECIFICATION}
     {Constructor Methods}
     PROCEDURE  Init  (ItemSize : WORD;
                       MaxItems : WORD);
     PROCEDURE  Clear;
     PROCEDURE  Push  (VAR Item   {UNTYPED};
                       ItemSize : WORD);
     PROCEDURE  Pop   (VAR Item  {UNTYPED};
                       ItemSize : WORD);
     {Selector Methods}
     FUNCTION   Is_Empty    : BOOLEAN;
     FUNCTION   Is_Full     : BOOLEAN;
     FUNCTION   Is_Status_Ok : BOOLEAN;
     FUNCTION   Items_In    : WORD;
     PROCEDURE  Show_Status (VAR F : TEXT  );
   END;


 QueueNodePtr  = ^QueueNode;

 QueueNode    =
   Object(StackNode) {QueueNode is an extension
                      of StackNode}
    {extended method}
     PROCEDURE SetNext ( {to} Ptr : QueueNodePtr );
   END;

 VisitProc    =  PROCEDURE (VAR Item);

 Queue        =
   Object(Stack) {Queue is an extension of Stack}
```

```
    {REPRESENTATION}
     Bottom       : QueueNodePtr;
    {SPECIFICATION}
     {Constructors}
     PROCEDURE  Init  (ItemSize : WORD;
                  MaxItems : WORD);
     PROCEDURE  Clear;

     PROCEDURE  EnQueue  (VAR Item   {UNTYPED};
                    ItemSize : WORD);
     PROCEDURE  DeQueue   (VAR Item  {UNTYPED};
                     ItemSize : WORD);
     {Iterator}
     PROCEDURE  Traverse (Visit : VisitProc);
    END;

TActiva=record
    Index:word;
   end;

TVActiva=record
     Index:word;
     Nombre:ShortString;
     LVP:word;
     VP:ShortString; {WVector;}
     i,j:word;
     cApar:byte;
     cM3:byte;
     cR:word;
     Ok:byte; {boolean;}
     Index2:word;{-----------------------------}
    end;

TRVcpm=record
    Index:word;
    i,j:word;
    Dur:real;{word;}
    iPx,tPx:real;{integer;}
    iLx,tLx:real;{integer;}
    hTotal,
    hLib,
    hInter:real;{integer;}
   end;

TRFic=record
    i,
    j:word;
```

```
     end;

TRC=record
    Index,
    i,
    j:word;
   end;

TRLin6=array[0..6] of ShortString;
TRLin8=array[0..8] of ShortString;
TRLin9=array[0..9] of ShortString;

TStr4=string[4];

TcParm=array[1..5] of string;

TVP=array[1..100] of word;
TGrupo=array[1..maxGpos] of ShortString;
TRGpos=record
      Cad:TStr4;
      Si:boolean;
     end;

OneActiva= ARRAY [1..1] OF TVActiva;
OneActivaPtr = ^OneActiva;

AVector = OBJECT
  MaxVectorSize,
  VSize : WORD;
  VecPtr : OneActivaPtr;
  VectorOpStatus : BOOLEAN;
  CONSTRUCTOR Init(MaxElem : WORD { input });
  DESTRUCTOR  Done;
  FUNCTION CurrentSize : WORD;
  PROCEDURE Store ( Index  : WORD; { input  }
                A      : TVActiva  { input  });
  PROCEDURE Recall( Index  : WORD; { input  }
            VAR  A      : TVActiva  { output });
  end;

OneCPM= array[1..1] of TRVcpm;
OneCPMPtr=^OneCPM;
TVcpm = OBJECT
  MaxVectorSize,
  VSize : WORD;
  VecPtr : OneCPMPtr;
  VectorOpStatus : BOOLEAN;
```

```
  CONSTRUCTOR Init(MaxElem : WORD { input });
  DESTRUCTOR  Done;
  FUNCTION CurrentSize : WORD;
  PROCEDURE Store ( Index  : WORD; { input  }
                A      : TRVcpm  { input  });
  PROCEDURE Recall( Index  : WORD; { input  }
            VAR  A      : TRVcpm  { output });
  end;



 OneGpo= ARRAY [1..1] OF ShortString;
 OneGpoPtr= ^OneGpo;

 GpoVector = OBJECT
  MaxVectorSize,
  VSize : Word;
  VecPtr : OneGpoPtr;
  VectorOpStatus : BOOLEAN;
  CONSTRUCTOR Init(MaxElem : WORD { input });
  DESTRUCTOR  Done;
  FUNCTION CurrentSize : WORD;
  PROCEDURE Store ( Index  : WORD; { input  }
                Gp      : ShortString  { input  });
  PROCEDURE Recall( Index  : WORD; { input  }
            VAR  Gp      : ShortString  { output });
  end;

 OneFic= ARRAY [1..1] OF TRFic;
 OneFicPtr= ^OneFic;

 TFicVector = OBJECT
  MaxVectorSize,
  VSize : Word;
  VecPtr : OneFicPtr;
  VectorOpStatus : BOOLEAN;
  CONSTRUCTOR Init(MaxElem : WORD { input });
  DESTRUCTOR  Done;
  FUNCTION CurrentSize : WORD;
  PROCEDURE Store ( Index  : WORD; { input  }
                Fic      : TRFic  { input  });
  PROCEDURE Recall( Index  : WORD; { input  }
            VAR  Fic      : TRFic  { output });
  end;

Var
 Gpo,VDura:GpoVector;
```

```
 JGpo:WVector;
 VActiva1:AVector;
 Vcpm:TVcpm;
 FicVector:TFicVector;
 VEstima:TVEstima;
 qHol,
 ColaFic,
 Cola1,
 cGpos:Queue;
 RIndex:TRIndex;
 JMax,nT:integer;
 DurPro:real;{integer;}
 cc32,
 flEncolar,
 fliP,
 flJMax:boolean;
 sTP,NomAr:string;
 sigm,
 Mu,Sigma:real;

implementation

CONSTRUCTOR WVector.Init(MaxElem : WORD { input });
BEGIN
  MaxVectorSize := MaxElem;
  VSize := 0;
  GetMem(VecPtr,MaxElem*SizeOf(word));
  VectorOpStatus := TRUE;
END;

DESTRUCTOR WVector.Done;
BEGIN
  MaxVectorSize := 0;
  VSize := 0;
  FreeMem(VecPtr,MaxVectorSize*SizeOf(word));
  VecPtr := NIL;
  VectorOpStatus := TRUE;
END;

FUNCTION WVector.CurrentSize: WORD;
BEGIN
  VectorOpStatus := TRUE;
  CurrentSize := VSize
END;

PROCEDURE WVector.Store(  Index  : WORD; { input }
                          X      : word  { input });
```

```
BEGIN
  IF Index>MaxVectorSize THEN BEGIN
    VectorOpStatus := FALSE;
    EXIT
  END;
  VecPtr^[Index] := X;
  IF Index>VSize THEN
    VSize := Index;
  VectorOpStatus := TRUE
END;

PROCEDURE WVector.Recall( Index : WORD; { input }
                 VAR X    : word { output });
BEGIN
  IF Index>VSize THEN BEGIN
    VectorOpStatus := FALSE;
    X := 0;
    EXIT
  END;
  X := VecPtr^[Index];
  VectorOpStatus := TRUE
END;

PROCEDURE WVector.Fill( FillValue : word;  { input }
                 NumElem   : WORD   { input });
VAR I : WORD;

BEGIN
  IF NumElem>MaxVectorSize THEN
    NumElem := MaxVectorSize;
  VSize := NumElem;
  FOR I := 1 TO NumElem DO
    VecPtr^[I] := FillValue;
  VectorOpStatus := TRUE
END;

PROCEDURE WVector.AddScalar(Scalar: word { input });

VAR I : WORD;

BEGIN
  FOR I:= 1 TO VSize DO
     VecPtr^[I]:=VecPtr^[I]+Scalar;
  VectorOpStatus:=TRUE
END;

PROCEDURE WVector.MultScalar(Scalar:word { input });
```

```
VAR I : WORD;

BEGIN
  FOR I:=1 TO VSize DO
     VecPtr^[I]:=VecPtr^[I]*Scalar;
  VectorOpStatus:=TRUE
END;

PROCEDURE WVector.AddVector(VectB:WVector { input });

VAR I : WORD;

BEGIN
  IF Vsize<>VectB.VSize THEN BEGIN
     VectorOpStatus:=FALSE;
     EXIT;
  END;
  FOR I:= 1 TO VSize DO
     VecPtr^[I]:=VecPtr^[I]+
              VectB.VecPtr^[I];
     VectorOpStatus:=TRUE
END;

FUNCTION WVector.MultVector(VectB:WVector { input }) : word;

VAR I:WORD;
   Sum:word;
   BEGIN
     IF VSize<>VectB.VSize THEN BEGIN
        VectorOpStatus:=FALSE;
        MultVector:=0;
        EXIT;
     END;
     SUM:=0;
     FOR I:=1 TO VSize DO
        Sum:= Sum+VecPtr^[I]*VectB.VecPtr^[I];
     VectorOpStatus:=TRUE;
     MultVector:=Sum { return function value }
   END;

CONSTRUCTOR TVEstima.Init(MaxElem : WORD { input });
BEGIN
  MaxVectorSize := MaxElem;
  VSize := 0;
  GetMem(VecPtr,MaxElem*SizeOf(word));
  VectorOpStatus := TRUE;
```

```
END;

DESTRUCTOR  TVEstima.Done;
BEGIN
  MaxVectorSize := 0;
  VSize := 0;
  FreeMem(VecPtr,MaxVectorSize*SizeOf(word));
  VecPtr := NIL;
  VectorOpStatus := TRUE;
END;

FUNCTION TVEstima.CurrentSize : WORD;
BEGIN
  VectorOpStatus := TRUE;
  CurrentSize := VSize
END;

PROCEDURE TVEstima.Store ( Index  : WORD; { input  }
             X      : TEstima  { input  });
BEGIN
  IF Index>MaxVectorSize THEN BEGIN
    VectorOpStatus := FALSE;
    EXIT
  END;
  VecPtr^[Index] := X;
  IF Index>VSize THEN
    VSize := Index;
  VectorOpStatus := TRUE
END;

PROCEDURE TVEstima.Recall( Index  : WORD; { input  }
             VAR  X      : TEstima  { output });
BEGIN
  IF Index>VSize THEN BEGIN
    VectorOpStatus := FALSE;
    X.tOp := 0;
    X.tMe := 0;
    X.tPe := 0;
    X.DurMe := 0;
    EXIT
  END;
  X := VecPtr^[Index];
  VectorOpStatus := TRUE
END;

PROCEDURE TVEstima.Fill(   FillValue : TEstima; { input }
               NumElem    : WORD  { input });
```

```
VAR I : WORD;

BEGIN
  IF NumElem>MaxVectorSize THEN
    NumElem := MaxVectorSize;
  VSize := NumElem;
  FOR I := 1 TO NumElem DO
    VecPtr^[I] := FillValue;
  VectorOpStatus := TRUE
END;

CONSTRUCTOR AVector.Init(MaxElem : WORD { input });
BEGIN
  MaxVectorSize := MaxElem;
  VSize := 0;
  GetMem(VecPtr,MaxElem*SizeOf(TVActiva));
  VectorOpStatus := TRUE;
END;

DESTRUCTOR AVector.Done;
BEGIN
  MaxVectorSize := 0;
  VSize := 0;
  FreeMem(VecPtr,MaxVectorSize*SizeOf(TVActiva));
  VecPtr := NIL;
  VectorOpStatus := TRUE;
END;

FUNCTION AVector.CurrentSize: WORD;
BEGIN
  VectorOpStatus := TRUE;
  CurrentSize := VSize
END;

PROCEDURE AVector.Store(  Index  : WORD; { input }
                          A      : TVActiva  { input });
BEGIN
  IF Index>MaxVectorSize THEN BEGIN
    VectorOpStatus := FALSE;
    EXIT
  END;
  VecPtr^[Index] := A;
  IF Index>VSize THEN
    VSize := Index;
  VectorOpStatus := TRUE
END;
```

```
PROCEDURE AVector.Recall( Index : WORD; { input }
                    VAR A    : TVActiva { output });
BEGIN
  IF Index>VSize THEN BEGIN
    VectorOpStatus := FALSE;
    {A := Nil; Corregir a una actividad nula}
    EXIT
  END;
  A := VecPtr^[Index];
  VectorOpStatus := TRUE
END;

CONSTRUCTOR TVcpm.Init(MaxElem : WORD { input });
BEGIN
  MaxVectorSize := MaxElem;
  VSize := 0;
  GetMem(VecPtr,MaxElem*SizeOf(TRVcpm));
  VectorOpStatus := TRUE;
END;

DESTRUCTOR TVcpm.Done;
BEGIN
  MaxVectorSize := 0;
  VSize := 0;
  FreeMem(VecPtr,MaxVectorSize*SizeOf(TRVcpm));
  VecPtr := NIL;
  VectorOpStatus := TRUE;
END;

FUNCTION TVcpm.CurrentSize: WORD;
BEGIN
  VectorOpStatus := TRUE;
  CurrentSize := VSize
END;

PROCEDURE TVcpm.Store(  Index  : WORD; { input }
                      A      : TRVcpm  { input });
BEGIN
  IF Index>MaxVectorSize THEN BEGIN
    VectorOpStatus := FALSE;
    EXIT
  END;
  VecPtr^[Index] := A;
  IF Index>VSize THEN
    VSize := Index;
  VectorOpStatus := TRUE
END;
```

```
PROCEDURE TVcpm.Recall( Index : WORD; { input }
                  VAR A    : TRVcpm { output });
BEGIN
  IF Index>VSize THEN BEGIN
    VectorOpStatus := FALSE;
    {A := Nil; Corregir a una actividad nula}
    EXIT
  END;
  A := VecPtr^[Index];
  VectorOpStatus := TRUE
END;

CONSTRUCTOR GpoVector.Init(MaxElem : WORD { input });
BEGIN
  MaxVectorSize := MaxElem;
  VSize := 0;
  GetMem(VecPtr,MaxElem*SizeOf(ShortString));
  VectorOpStatus := TRUE;
END;

DESTRUCTOR GpoVector.Done;
BEGIN
  MaxVectorSize := 0;
  VSize := 0;
  FreeMem(VecPtr,MaxVectorSize*SizeOf(ShortString));
  VecPtr := NIL;
  VectorOpStatus := TRUE;
END;

FUNCTION GpoVector.CurrentSize: WORD;
BEGIN
  VectorOpStatus := TRUE;
  CurrentSize := VSize
END;

PROCEDURE GpoVector.Store(  Index  : WORD; { input }
                    Gp     : ShortString  { input });
BEGIN
  IF Index>MaxVectorSize THEN BEGIN
    VectorOpStatus := FALSE;
    EXIT
  END;
  VecPtr^[Index] := Gp;
  IF Index>VSize THEN
    VSize := Index;
  VectorOpStatus := TRUE
```

```
END;

PROCEDURE GpoVector.Recall( Index : WORD; { input }
                  VAR Gp    : ShortString { output });
BEGIN
  IF Index>VSize THEN BEGIN
    VectorOpStatus := FALSE;
    {A := Nil; Corregir a una actividad nula}
    EXIT
  END;
  Gp := VecPtr^[Index];
  VectorOpStatus := TRUE
END;

CONSTRUCTOR TFicVector.Init(MaxElem : WORD { input });
BEGIN
  MaxVectorSize := MaxElem;
  VSize := 0;
  GetMem(VecPtr,MaxElem*SizeOf(TRFic));
  VectorOpStatus := TRUE;
END;

DESTRUCTOR TFicVector.Done;
BEGIN
  MaxVectorSize := 0;
  VSize := 0;
  FreeMem(VecPtr,MaxVectorSize*SizeOf(TRFic));
  VecPtr := NIL;
  VectorOpStatus := TRUE;
END;

FUNCTION TFicVector.CurrentSize: WORD;
BEGIN
  VectorOpStatus := TRUE;
  CurrentSize := VSize
END;

PROCEDURE TFicVector.Store(  Index  : WORD; { input }
                    Fic     : TRFic  { input });
BEGIN
  IF Index>MaxVectorSize THEN BEGIN
    VectorOpStatus := FALSE;
    EXIT
  END;
  VecPtr^[Index] := Fic;
  IF Index>VSize THEN
    VSize := Index;
```

```
  VectorOpStatus := TRUE
END;

PROCEDURE TFicVector.Recall( Index : WORD; { input }
                   VAR Fic    : TRFic { output });
BEGIN
  IF Index>VSize THEN BEGIN
    VectorOpStatus := FALSE;
    {A := Nil; Corregir a una actividad nula}
    EXIT
  END;
  Fic := VecPtr^[Index];
  VectorOpStatus := TRUE
END;

  { ************************************************* }
  { StatusType Method                          }
  { ************************************************* }
  PROCEDURE  StatusType.Reset;
    BEGIN
      OverFlow               := FALSE;
      UnderFlow              := FALSE;
      Object_MisMatch        := FALSE;
      Dynamic_Memory_Exhausted := FALSE;
      General_Failure        := FALSE
    END {StatusType.Reset};
  { ************************************************* }
  { StackNode Methods                          }
  { ************************************************* }
  PROCEDURE StackNode.Init (  VAR Captured_Info; {untyped}
                          Info_Size : WORD;
                          Next_Node : StackNodePtr) ;
    BEGIN
      GetMem (Info, Info_Size);
      Move (BYTE(Captured_Info), BYTE(Info^), Info_Size);
      Next := Next_Node
    END {StackNode.Init};
  { ************************************************* }
  PROCEDURE StackNode.Yield_Info ( VAR Yielded_Info; {untyped}
                            Info_Size : WORD     );
    BEGIN
      Move (BYTE(Info^), BYTE(Yielded_Info), Info_Size)
    END {StackNode.Yield_Info};
  { ************************************************* }
  PROCEDURE StackNode.Clear ( Info_Size : WORD );
    BEGIN
      FreeMem (Info, Info_Size)
```

```
  END {Stack_Node.Clear};
{ ************************************************** }
{ Stack Methods                                      }
{ ************************************************** }
PROCEDURE  Stack.Init  (ItemSize : WORD;
                        MaxItems : WORD);
  BEGIN
    Top        := NIL;
    Size       := 0;
    Last_Op_Ok := TRUE;
    MaxSize    := MaxItems;
    InfoSize   := ItemSize;
    Status.Reset
  END { Stack.Init };
{ ************************************************** }
PROCEDURE  Stack.Clear;
  VAR
    Ptr  : StackNodePtr;
  BEGIN
    WHILE Top <> NIL DO BEGIN
      Ptr := Top;
      Top := Top^.Next;
      Dec (Size);
      Ptr^.Clear (InfoSize);
      Dispose (Ptr)
    END {While};
    Status.Reset;
    Last_Op_Ok := TRUE
  END {Stack.Clear};
{ ************************************************** }
FUNCTION   Stack.Is_Empty : BOOLEAN;
  BEGIN
    Is_Empty := Size = 0
  END {Stack.Is_Empty};
{ ************************************************** }
FUNCTION   Stack.Is_Full  : BOOLEAN;
  BEGIN
    Is_Full := Size >= MaxSize
  END {Stack.Is_Full};
{ ************************************************** }
FUNCTION   Stack.Is_Status_Ok  : BOOLEAN;
  BEGIN
    Is_Status_Ok := Last_Op_Ok
  END {Stack.Is_Status_Ok};
{ ************************************************** }
FUNCTION   Stack.Items_In  : WORD;
  BEGIN
```

```
    Items_In := Size
  END {Stack.Items_In};
{ *************************************************** }
PROCEDURE  Stack.Push  (VAR Item   {UNTYPED};
                 ItemSize : WORD);
  VAR
    NewNodePtr : StackNodePtr;
  BEGIN
    Status.Reset;
    Last_Op_Ok := FALSE;
    IF (Size >= MaxSize)THEN
      Status.OverFlow := TRUE
    ELSE IF (ItemSize <> InfoSize) THEN
      Status.Object_MisMatch := TRUE
    ELSE BEGIN
      Last_Op_Ok := TRUE;
      New (NewNodePtr);
      NewNodePtr^.Init ( {inserting} Item, {of} ItemSize, Top);
      Top := NewNodePtr;
      Inc (Size);
    END {If}
  END {Stack.Push};
{ *************************************************** }
PROCEDURE  Stack.Pop   (VAR Item   {UNTYPED};
                 ItemSize : WORD);
  VAR
    TrashNodePtr : StackNodePtr;
  BEGIN
    Status.Reset;
    Last_Op_Ok := FALSE;
    IF NOT (Size > 0) THEN
      Status.UnderFlow := TRUE
    ELSE IF (ItemSize <> InfoSize) THEN
      Status.Object_MisMatch := TRUE
    ELSE BEGIN
      Last_Op_Ok := TRUE;
      TrashNodePtr := Top;
      Top := Top^.Next;
      Dec (Size);
      TrashNodePtr^.Yield_Info ( Item, {of} ItemSize);
      TrashNodePtr^.Clear (ItemSize);
      Dispose (TrashNodePtr)
    END {If};
  END {Stack.Pop};
{ *************************************************** }
PROCEDURE  Stack.Show_Status (VAR F : TEXT  );
  BEGIN
```

```
    Write (F, 'OBJECT STATUS:  ');
    IF Last_Op_Ok THEN
     WriteLn (F,
      'Last constructor was successful . . .')
    ELSE IF Status.OverFlow THEN
     WriteLn (F,
      'Attempted an insertion into a full structure')
    ELSE IF Status.UnderFlow THEN
     WriteLn (F,
      'Attempted a deletion from an empty structure')
    ELSE IF Status.Object_MisMatch THEN
     WriteLn (F,
      'Attempted an insertion/deletion with wrong type object')
    ELSE IF Status.Dynamic_Memory_Exhausted THEN
     WriteLn (F,
      'Heap exhausted -- attempted insertion ignored')
    ELSE IF Status.General_Failure THEN
     WriteLn (F,
      'General failure')
    {EndIf}
  END {Stack.Show_Status};
{ ************************************************** }
{ ************************************************** }

{ ************************************************** }
{ QueueNode Method -- extends methods of StackNode      }
{ ************************************************** }
PROCEDURE QueueNode.SetNext ( {to} Ptr : QueueNodePtr );
  BEGIN
    Next := Ptr
  END {QueueNode.SetNext};

{ ************************************************** }
{ Queue Methods                              }
{ ************************************************** }
PROCEDURE  Queue.Init  (ItemSize : WORD;
                 MaxItems : WORD);
  BEGIN
    Stack.Init (ItemSize, MaxItems);
    Bottom := NIL
  END {Queue.Init};
{ ************************************************** }
PROCEDURE Queue.Clear;
  BEGIN
    Stack.Clear;
    Bottom := NIL
  END; {Queue.Clear}
```

```
{ ************************************************** }
PROCEDURE  Queue.EnQueue  (VAR Item   {UNTYPED};
                   ItemSize : WORD);
  VAR
    NewNodePtr : QueueNodePtr;
    {NewElemPtr : POINTER;}
  BEGIN
    Status.Reset;
    Last_Op_Ok := FALSE;
    IF (Size >= MaxSize) THEN
      Status.OverFlow := TRUE
    ELSE IF (ItemSize <> InfoSize) THEN
      Status.Object_MisMatch := TRUE
    ELSE BEGIN
      Last_Op_Ok := TRUE;
      NEW (NewNodePtr);
      NewNodePtr^.Init ( Item, ItemSize, NIL );
      IF (Size = 0) THEN
        Top    := NewNodePtr
      ELSE
        Bottom^.SetNext ( {to} NewNodePtr )
      {EndIf};
      Bottom := NewNodePtr;
      INC (Size)
    END {If}
  END {Queue.EnQueue};
{ ************************************************** }
PROCEDURE  Queue.DeQueue   (VAR Item   {UNTYPED};
                   ItemSize : WORD);
  BEGIN
    Pop (Item, ItemSize);
    IF Size = 0 THEN
      Bottom := NIL
    {EndIf}
  END {Queue.DeQueue};
{ ************************************************** }
PROCEDURE Queue.Traverse (Visit : VisitProc);
  VAR
    SaveTop : POINTER;
  BEGIN
    SaveTop := Top;
    WHILE Top <> NIL DO BEGIN
      Visit ( BYTE(Top^.Info^) ); {cheats to look at node}
      Top := Top^.Next
    END {While};
    Top := SaveTop;
    Status.Reset;
```

```
    Last_Op_Ok := TRUE
   END {Stack.Clear};
 { ************************************************** }
 { ************************************************** }

end.
```